Appendix S1

Marine Technical Methods Project Instructions

Building a Temperature-Depth Logger

Table of Contents

The key skills, theoretical background, and applications of technology that we will cover in this course have been integrated into a hands-on project that will span the first half of the course.

GOALS OF PROJECT

Through building a temperature-depth logger, students will:

- Become familiar with different types of sensors
    - Temperature, pressure
    - Analog vs digital – advantages and disadvantages
    - Sensor outputs – current vs voltage
- Gain familiarity with using microcontrollers to sense and collect environmental data
    - Software, Electrical hardware, Mechanical hardware
- Learn or improve technical skills
    - Use of multimeter
    - Soldering
    - Drilling and Tapping
- Learn or improve programming skills
    - Breaking problems into discrete steps
    - Arduino programming
    - MATLAB – data analysis
- Be familiar with electronics theory
    - Circuits and Ohm's law
    - Resistors
    - Voltage dividers
- Learn to design and connect circuit components using breadboards and perf boards
- Use web resources to learn about electronics
    - Background information: https://www.electronics-tutorials.ws/
    - Specific sensors and boards: https://www.adafruit.com/
    - Lots of other tutorials!
- Be introduced to challenges of using electronics in marine environments
    - Steps for a standalone environmental data logger
    - Waterproofing instruments
- Build an instrument to collect data!
- Analyze data from multiple instruments

HOW THIS PROJECT WORKS

Our goal is for you to learn how the process of doing science works, so you will be given instructions but will be expected to read websites, look things up on the internet, and think through problems and come up with your own solutions. This is not a cookbook lab. We also anticipate that students will come in to this course with different backgrounds and will work at different paces. There are optional exercises if you find yourself ahead of the timeline (or feel free to suggest your own ideas!), and if you find yourself behind the timeline, don't panic. That said, it's very important that you come to class prepared, having read the project instructions and completed the before class questions, so you are ready to work during class time.

PROJECT OVERVIEW

PART 1 – TEMPERATURE SENSOR

Part 1.1: Learn about temperature sensors

- Design and connect electrical components using a breadboard
- Learn to use a multimeter with electrical circuits
- Program software to measure temperature (with thermistors and digital sensors) using an Arduino microcontroller
- Prototype circuit design

Part 1.2: Start building the instrument – build the hardware

- Put your circuit on a perf board
- Mount it on acrylic to put in the waterproof housing

Part 1.3: Logging your data – SD card shield

- Solder headers on the SD card shield so you can connect it to the Arduino microcontroller
- Set up the Real Time Clock (RTC) on your SD card shield
- Write the software to log data to the SD card
- MATLAB tutorial so you're ready to plot data
- Log data overnight and make a graph in MATLAB

PART 2 – PRESSURE SENSOR

Part 2.1: Learn about pressure sensors

- Background question to answer before class
- Design and connect electrical components to measure pressure with a 4-20mA output pressure sensor
- Write software to measure pressure using an Arduino microcontroller

Part 2.2: Continue building the instrument

- Add pressure sensor to perf board with temp sensor

Part 2.3: Software and calibration

- Calibrate the pressure sensor
- Integrate the pressure sensor code into software for the temperature sensor (time permitting)

PART 3: PREPARING FOR FIELD DEPLOYMENT

Part 3.1: Finish the software

- Plan coding for temperature calculation before class
- Calculate temperature in Arduino code
- Calibrate the sensor / check temperature calculation

Part 3.2: Powering your instrument

- Determine current draw of instrument to do battery calculations
- Learn about batteries and calculate battery requirements
- Add a battery pack to your instrument

Part 3.3: Waterproof your instrument

- Put instruments in waterproof housing
- Go through checklist – double check so no flooding!
- Deploy off dock and collect data!

PART 4: FIELD DEPLOYMENT AND DATA ANALYSIS

Part 4.1: Analyze your data from dock deployment

- Import data and plot data in MATLAB
- Compare it to ARCOS weather station data – download, import, and plot data

Part 4.2: Modify instruments to be castable, collect data and compare to CTD

- Import data and plot in MATLAB
- Plot CTD data in MATLAB
- Interpret data

Part 4.3: Lessons learned and next steps

WORKING TOGETHER VERSUS PLAGIARISM

You are strongly encouraged to work together in this class. Teaching and learning from your peers is a very good way of learning and reinforcing material. Keep in mind as you work together that the process of problem solving is a critical step in learning, so don't just share the answer, talk about the process of getting to the answer. In coding, the line between helping someone and plagiarism is a bit gray, especially since we are using code from the Arduino example library and the internet. In this class, showing a classmate your code while you discuss it is encouraged. Copying and pasting lines of code and sending to a classmate or sending entire sketches of code or photos of code, even if meant to teach or help, cross the line into plagiarism and is not allowed. If you're unsure about what is helping and what is plagiarism (this is not always clear!), please ask!

Part 1: Temperature Sensor

OVERVIEW

Part 1 is expected to take the first two weeks of class.
Part 1.1: Learn about temperature sensors

- Design and connect electrical components using a breadboard
- Learn to use a multimeter with electrical circuits
- Program software to measure temperature (with thermistors and digital sensors) using an Arduino microcontroller
- Prototype circuit design

Part 1.2: Start building the instrument – build the hardware

- Put your circuit on a perf board
- Mount it on acrylic to put in the waterproof housing

Part 1.3: Logging your data – SD card shield

- Solder headers on the SD card shield so you can connect it to the Arduino microcontroller
- Set up the Real Time Clock (RTC) on your SD card shield
- Write the software to log data to the SD card
- MATLAB tutorial so you're ready to plot data
- Log data overnight and make a graph in MATLAB

REPORT GRADING

Questions 1-14: 55 pts
MATLAB tutorial: 15 pts
Graph at end: 30 pts

PART 1.1: LEARN ABOUT TEMPERATURE SENSORS

BEFORE CLASS

BACKGROUND READING

Please read through this information before class. Don't worry about understanding all of it – you will likely need to refer back to these sites during class.
Introduction to sensors, actuators, and transducers; analog vs. digital sensors:
https://www.electronics-tutorials.ws/io/io_1.html

Different types of temperature sensors: thermostats, thermistors, RTDs, thermocouples
https://www.electronics-tutorials.ws/io/io_3.html

We are going to measure temperature with a thermistor, and will use a voltage divider circuit to measure voltage. Before you start learning about voltage divider circuits, start with an introduction to circuits and ohms law – this should be at least vaguely familiar from your intro to physics course, but don't worry if you don't remember much:
https://www.electronics-tutorials.ws/dccircuits/dcp_1.html
https://www.electronics-tutorials.ws/dccircuits/dcp_2.html

Next read about voltage dividers:
https://www.electronics-tutorials.ws/dccircuits/voltage-divider.html

If you've never soldered before, read/watch this:
https://www.makerspaces.com/how-to-solder/

CALCULATIONS TO DO BEFORE CLASS

We are going to use this equation (eq. 1) several times during class:

$$V_{OUT} = V_{IN} \frac{R_2}{R_1 + R_2}$$

Here, Vout is the voltage out of our circuit, Vin is the voltage input (in our case, it will be 5V), R_2 is the resistance of the resistor closest to ground in the circuit, and R_1 is the resistance of the resistor closest to the voltage input. You are going to need to do some algebra and solve this equation first for R_2 (so R_2 = …) and then for R_1. Do these calculations before coming to class.

MATLAB TUTORIAL

You will need to complete this **MATLAB tutorial** (~2 hrs) before plotting your data and turn in the certificate with your project report:
https://www.mathworks.com/learn/tutorials/matlab-onramp.html

The tutorial should take you about 2 hours and covers:

- How to enter commands and start to use MATLAB – you won't know where to start if you don't do this part
- How to deal with arrays – we will import data as a table, which is a bit more complicated than an array but you will need to subsample parts of your data using the indexing techniques in lesson 5
- A few basic functions and how to get help on MATLAB commands – note here that google can sometimes be more helpful than the MATLAB documentation, especially if you don't know exactly what the function you're looking for is called… just google, e.g., "MATLAB random number"
- How to make plots – we will be plotting our data and will need to change the colors and symbols and line types
- How to import data – this is especially tricky with the date and time formats that we will be using; we will be using tables and there's a brief introduction to how tables work
- A few practice examples – these are useful in showing you the capabilities of mat MATLAB lab and applying what you've learned

Once you're done, save your certificate as a pdf and turn it in.

If you've used MATLAB but not extensively, I recommend doing the tutorial as a refresher – it will take less than 2 hours and you might pick up some new tips. If you are proficient in MATLAB and would like to be excused from doing the tutorial, ask at least 24 hrs before the assignment deadline.

IN CLASS

PARTS LIST

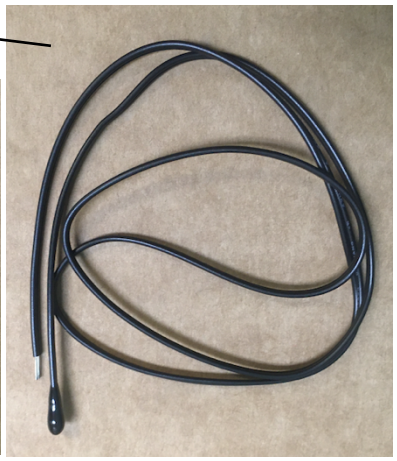
Most of these parts will be in your toolbox – feel free to use wires and other components as you need them and ask if you're missing anything. Take some time to look through the class supplies to familiarize yourself with the available tools.

General supplies

- Arduino, USB cable
- Breadboard
- Multimeter
- Jumper wires for breadboard
- 24gage stranded wire for perf board (red for power, black for ground, other colors)
- Screw terminals (and sharpies to label them)
- Resistors
- Drill and drill bits

Project-specific supplies

- Adafruit SD card shield - https://www.adafruit.com/product/1141
  - SD card
  - Coin cell battery (CR1220 12mm diam 3V lithium)
- 10 kΩ thermistor (use 1st)
- Potentiometer
- Adafruit 10K Precision Epoxy Thermistor - 3950 NTC - https://www.adafruit.com/product/372
- Dallas 1-wire Temperature probe (DS18B20)
- Prototyping / Perf board
- Acrylic to mount boards (must fit inside acrylic housing)
- 4-40 standoffs, 4-40 screws, 4-40 tap, #43 drill bit, drill; for mounting boards
- Transfer punch for mounting boards
- Ice water in beaker
- Thermometer to measure room temperature
- AC/DC power converter (12V, 2A)



FIRST LEARN TO SOLDER

We are going to start by soldering the headers onto the SD card shield that we will be using to log data. This is a good project to learn to solder. We will use an Adafruit SD card shield – they tend to have very helpful information on their website so this is a good resource for a lot of projects you might do. Here's the site for the SD card shield:
https://learn.adafruit.com/adafruit-data-logger-shield

We will do a soldering demo in class. You'll solder the headers on the data logger shield and will also need to solder 4 screw terminals on to the data logger shield and wire them to 5V power, ground, and analog pins A0 and A1 – we will use these later in the project. Screw terminals are very useful because you can secure your wires to your board but still easily remove them (to disconnect the different parts of your instruments, swap out parts that are bad, etc.) Once you're done soldering, put the SD card shield aside – you will need it later but not right away.

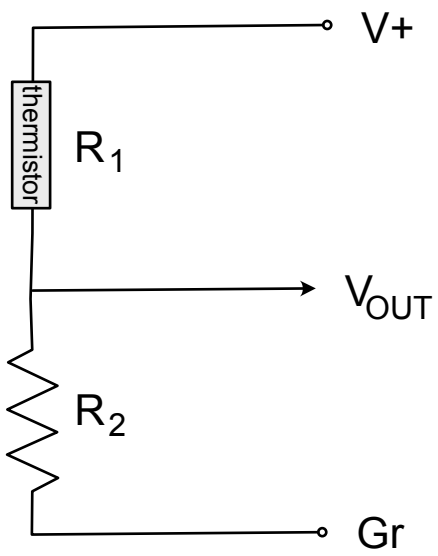
1.1 WIRING YOUR TEMPERATURE SENSOR

We are going to start out by rigging up a basic thermistor temperature sensor and reading the temperature with an Arduino.

STEP 1 – WIRE THE CIRCUIT ON A BREADBOARD AND CHECK IT WITH A MULTIMETER

First hook up the **multimeter**… Your multimeter is what you will use to troubleshoot/evaluate our electronic constructions! You have two types of probes: gator clips and pointy probes. Feel free to interchange them so they are the most convenient for you. The black lead is usually the low side of the circuit (i.e., closer to ground, or the side with the lowest potential energy) and should be plugged into the "COM" port of your multimeter. The red lead can be plugged into either of the remaining two ports, depending on your application. For now, you should use the right port ("VΩHz") which will allow you to measure the voltage (V), resistance (Ω), or frequency (Hz) of your circuit. You can also use this port to measure small currents (mA or μA). You will need to change it to the left port if you want to measure larger currents (A). NOTE: You will blow a fuse (i.e. damage) your multimeter if you attempt to measure large currents with the probe in the right port! ALSO NOTE: Red and black can be interchanged, but it's a good idea to follow convention when possible!

You will wire your thermistor on your breadboard following this circuit:



This circuit is very common in electronics and is called a **voltage divider**. They are used to take an input voltage, and based on the combination of resistors, output a proportionally lower voltage. We will cover the specifics of how this is accomplished later in this project.

Figure was modified from https://www.electronics-tutorials.ws/io/thermistors.html to match our set-up. V+ is voltage in, Vout is signal voltage out, Gr is ground, R_1 is thermistor, R_2 is resistor. Note that you can set the circuit up with the thermistor and resistor switched from this version – that seems to be the more common set-up but this version is a bit more intuitive for our intro exercise.

To understand this circuit, you'll need to know **Ohm's Law:**

V=IR

V = voltage (V), I = current (A, amps), R = resistance (Ω, ohms)

You'll need a resistor and a thermistor – we'll use 10 kΩ thermistors (so R_1 = 10 kΩ). Check your thermistor with a multimeter to confirm that it's close to 10 kΩ. Turn the multimeter on to resistance (look for the Ω (which stands for ohms) symbol) and touch the red and black leads to the two ends of the thermistor. Note that the multimeter will indicate if the circuit is open, e.g. with "0L."

1. What happens to the resistance value of your thermistor if you heat it (pinch it with your fingers to use your body heat to warm it up)? As concisely as you can, describe the resistance output of the thermistor in response to temperature changes? (2 pts)

To figure out what size resistor to use in our circuit to measure temperature with the thermistor, consider that we are using the Arduino to apply the voltage and the Arduino output is 5V. We are going to use the analog input, A0, which reads between 0 and 5V. At a first approximation, we want our output signal to fall in the middle of the range that the Arduino reads, so ~2.5V is a good target. Depending on our application, we might choose a different target – for what application might you choose a higher or lower target value?

2. Use this equation (eq. 1) to calculate the value resistor we should use (R_2) for our 10 kΩ thermistor (R_1):

$$V_{OUT} = V_{IN} \frac{R_2}{R_1 + R_2}$$

   Note that V_{IN} (or V+ in the circuit drawing above) is the voltage coming in to the circuit (but out of the Arduino!), and V_{OUT} is the voltage coming out of the voltage divider (and going in to the Arduino). Show your work (3 pts) and give the resistor value (2 pts).

Find the right resistor and check with your multimeter to make sure it has the right resistance value (note that the colors can be hard to distinguish). Use your Arduino and breadboard to make the circuit to measure temperature using the thermistor. Hook up the V+ and Gr connections of your circuit to the Arduino, and plug the Arduino into your computer. The program doesn't matter yet… we are simply using the computer to power the circuit through the Arduino's 5V regulator for now. Next, put your Multimeter in VDC (voltage, direct current) mode and touch the black lead to a Gr and the red lead to V+.

3. (a) What do you expect the reading to be? What reading does your multimeter return? What are the Units? (2 pts)

   (b) Next connect the red lead (high side) to the signal (Vout) and the black lead to Gr. What do you expect the reading to be? What is your new reading from the multimeter? (don't forget units!) (2 pts)

4. Use the equation above to calculate the output voltage at room temperature if we used a resistor (R_2) with greater (100 kΩ) resistance. What about lower (1000 Ω) resistance? Unplug your Arduino, switch the resistor, and plug it back in and check your predictions with the multimeter. Compare your measured to predicted values. (4 pts)

5. Go back to equation 1 - If you had a 100 kΩ thermistor, what size resistor would you use to achieve the same goal? Would your output voltage change (from the 10 kΩ thermistor and resistor that you wired up)? (2 pts)

STEP 2 - NOW IT'S TIME TO CODE!

The Arduino Software has lots of example programs (sketches) – depending on your application, it's often faster/easier to start with one of these examples and then modify the code to suit your needs rather than write the code from scratch. We are going to use serial communication to talk to our Arduino with our computer so we can see the Arduino output on our monitor. Here's a good intro to serial communication:
https://www.ladyada.net/learn/arduino/lesson4.html

It's a bit dated and not totally on topic so just skim it and make sure you get the gist of what we mean when we use the term "serial."

Back to the Arduino software, we will start with the example AnalogInOutSerial, which you can find under File -> Examples -> 03.Analog in the Arduino Software. Open this file, read through it, determine where you should plug in Vout from your circuit, and then upload the program to your Arduino to stream your data. Open the serial output window (click the button in the upper right corner) and look at the numbers. Hold the thermistor between your fingers and see how the numbers change.

6. What do the numbers you are seeing mean? Do they have units? If so, what are the units, and if not, how do they correspond to some real units? What is the highest (maxed out) the numbers can go? What does the Max number represent (in terms of capabilities of the Arduino)? (5 pts)

Now that your code is working, save the sketch file under a new name (e.g., TempLog) in a folder for the class. Now you're ready to **start modifying the code**.

Note that the code takes an analog input and then writes an analog output – read through the code and figure out which lines do each of those parts. We do not need the analog output, so delete those lines and then check to make sure your code still works. [Note: it's a good idea when you are programming to change one thing at a time and then check to make sure the code works before you change the next thing. When deleting things, it's a good idea to first comment out a line of code (with "//") and then check it before deleting it completely.]

Arduino (like many other programming languages) has different formats for numbers:

Float – has decimals, e.g., 1.045
Int – integers that do not have decimals, e.g., 1

You want to output a number with decimal places for voltage so will need to define your variable as "float" rather than "int" in the first part of the code. Arduino will convert "float" back to "int" if you don't include a decimal in your math (likely to save memory), so when you define the variable make sure you use a decimal place (e.g., "0.00" rather than "0"). Look up the Arduino help for "float" and "int" if this isn't totally clear (or ask!) – it's a source of many errors in code!

Next, convert your "sensorValue" to Volts: to do this, you'll need to create a new variable (e.g., "readVolt") and then write an equation that converts "sensorValue" to this new variable. So what does this mean?

Step 1: First you need to create a variable and define what kind of variable it is (int, float, etc.). Recall from the starter kit exercises that every Arduino script must have two functions in it: the void set-up and void loop. Notice how before the void setup function in the AnalogInOutSerial code, there are several variables that are defined. This is where you should define your new variable:

float readVolt = 0.00; //define variable to calculate voltage (V)

This line of code translates to

> *I am defining a new variable that is a 'float' and its name is 'readVolt.' The value I want this to start at is 0, but I want it to have decimal places (not be an integer) so I have to tell Arduino "0.00" not just "0." ";" means this is the end of my line of code and "//" means the start of my annotation that explains what this variable is so I don't forget what I did and also includes the units of that variable, which are volts.*

Note that there are two kinds of variables: variables that have a constant value (and start with *const* in Arduino language) and variables that have values that change (and don't start with *const*). Also, if you're going to use a variable in both the setup and the loop, you should define it at the beginning (this is a universal variable). If you define a variable within the setup, you can only use it in the setup (probably not what you want). But if you're only using a variable in the void loop, it's fine to define it there.

Step 2: Write the equation as "readVolt = …;" This goes in the void loop function right after you have read the *sensorValue* (since your *sensorValue* is an input in your *readVolt* equation). It's a good idea to first write out the equation you will need to convert the sensor value variable to volts ON PAPER before thinking about how to write it in Arduino coding language. Think about the range of possible "sensor value" numbers and the range of voltages in the Arduino analog input. [If you're not confident that your equation is right, ask! Or better yet, just try it and see if you get the an output that matches what you measure with your multimeter.]

Step 3: Tell your Arduino code to print out the new variable value to the serial monitor so you can see it. You probably want this to print after your *sensorValue*, so add your code there. You should look up the difference between *Serial.print* and *Serial.println* and make sure you're using the right one.

If you get stuck, it might help to read up on the *analogRead* function in Arduino:
https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/

Remember to annotate your code (using "//" before your annotations) with a description of what each line does and the units of each variable.

7. Why do you think you use variables in programming? When you define your variables, you define some as "const float" and some as "float" – how do you decide which one to use? (3 pts)

8. Next create a new variable in your code, "thermResistance", and convert your output value to the resistance of your thermistor. You'll need to use eq. 1 and rearrange it to solve for R_1 (see the circuit figure above). Write the equation in equation form before adding it to your code (you should have done this for homework). [include both equation and line of code in your answer here] (5 pts)

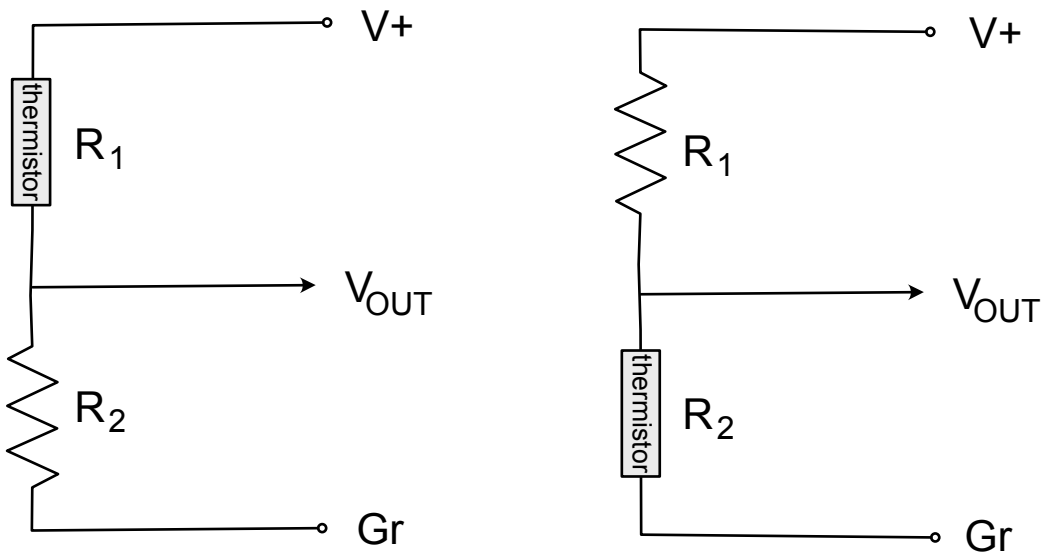
Recall how the resistance of your thermistor changes with temperature, and make sure your calculated resistance is (a) in the right ballpark of the resistance of your thermistor (remember the rating is the resistance at 25 °C), and (b) is changing in the right direction as you heat and cool your thermistor.

9. Check your resistance with your multimeter (you'll need to remove the thermistor from the circuit to do this). Write down the value and compare it to the calculated values you are seeing on your serial output window. How close are these values to each other and to the rated resistance of your thermistor? Your thermistor is rated to an accuracy of 1% - is this consistent with your numbers? If you needed a more accurate measure of temperature, what might you do to improve the accuracy of your measurement? (There are multiple right answers to this question!) (5 pts)

The thermistor we used is very low-cost and looks like a resistor (to drive home the idea that thermistors are variable resistors), but it is not particularly field-deployable. You can buy essentially the same thermistor that is more robust and has longer wires for only a couple extra dollars:
https://www.adafruit.com/product/372

Note that the temperature range of this sensor is -55 to 125 °C, but they recommend not heating above 105 °C. The Adafruit website is a great reference for DIY technology – not only do they have lots of boards and sensors, but they also have a lot of good tutorials and background information. There is an Adafruit tutorial to convert your Arduino readings from your sensor to resistance, but note that the Adafruit tutorial starts by setting up the circuit with the thermistor and resistor switched from our circuit:
https://learn.adafruit.com/thermistor/using-a-thermistor

Two options for wiring the thermistor circuit:



The circuit on the left gives a more intuitive Vout value and is what we used. The circuit on the right seems to be more broadly used. Both are fine, you just have to keep track of which one you're using and make sure you don't mix up R_1 and R_2 in your calculations from eq. 1.

10. Hook up the Adafruit thermistor following the circuit on the left (what we used for the other thermistor) and use your code to check the readings. What happens to the voltage when you heat the thermistor? Next switch the circuit to match the circuit diagram on the right – what is the voltage out at room temperature and how does it change when you heat the thermistor? For the first circuit, the thermistor is a "pull up resistor," whereas in the second circuit the thermistor is a "pull down resistor" – explain in your own words what these terms mean. (5 pts)

STEP 3 – LEARN MORE ABOUT SENSORS

To better understand how variable resistors work and the broad application of the voltage divider in sensors, we will look at a different kind of variable resistor. **Potentiometers** (also called Pots) are commonly used variable resistors that rely on a mechanical change to vary their resistance (caused by turning a knob) rather than a chemical change in resistance (caused by temperature) as seen with your thermistor.

11. The thermistor circuit you just finished should have 3 connections to your Arduino. Describe what these connections do in your own words. Notice that your potentiometer has 3 pins. Use a multimeter to figure out the identity (or function) of each of those 3 pins. Where on the Arduino do you think these pins should go? Include a circuit diagram, like the one above, in your answer, and annotate what the 3 wires are and how you connect them to your Arduino. (4 pts)

Put your thermistor aside and use your breadboard to connect the potentiometer to your Arduino. Use the same code that you used for the thermistor to get data from the potentiometer. Turn the knob and make sure it changes your data.

12. How do the changes in the readings from the potentiometer circuit compare to the changes you saw with the thermistor circuit? What are the units of the numbers you are reading from the potentiometer circuit? Explain the term "variable resistor" in your own words. (3 pts)

We will compare this with another type of temperature sensor, a **Dallas 1-wire Temperature probe (DS18B20)**. This is a DIGITAL sensor, and will be a little different from the ANALOG thermistor. As with most sensors, you can find a good tutorial on how to set this sensor up: https://lastminuteengineers.com/ds18b20-arduino-tutorial/

With the thermistor, we used a resistor in a voltage dividing circuit to output a reduced voltage in proportion to a change in resistance. We then read the voltage with the analog input of our Arduino. The DS18B20 is a digital sensor, which means it has a bunch of fancy electronics inside and outputs temperature through one wire that goes into a DIGITAL input pin on the Arduino. The interpretation of the data is a bit less intuitive, but there are Arduino libraries with examples to help you set it up. You'll need to install two libraries, "DallasTemperature" and "OneWire" which you can do with the Library Manager in Arduino (Tools -> Manage Libraries).

Put aside your thermistor sensor and circuit and the potentiometer circuit before you start wiring the Dallas Temperature probe. Follow the instructions on the Last Minute Engineers tutorial to set up the wiring for the Dallas Temperature probe on your breadboard and Arduino, then copy and paste the Arduino code from the tutorial (note the copy button at the top right of the box of code) into a new Arduino sketch. Read the temperature in your serial output.

When designing data loggers (or most things…) you will need to identify the best sensor (or material, or mechanism, etc…) for your application. This means identifying advantages and disadvantages of different sensor types and using that info to make an informed design decision. We will revisit this concept frequently over the next few weeks.

13. In your own words, what is the difference between analog and digital sensing? What are some advantages and disadvantages of each? Do you have a personal preference for one or the other, or can you think of contexts in which one might be preferable over the other? Why? (5 pts)

We are going to use the **Adafruit thermistor probe** for our field instrument, so put away the DS18B20 1-wire probe. We chose this probe because it fits better in the thermowell housing that we have, but there are lots of reasons for choosing one over the other.

14. What other factors should we consider in choosing a temperature probe for our instrument? (3 pts)

Note: We will convert ThermResistance to temperature (°C) in Part 3.1 – for now, we will work with our temperature data in resistance or volts.

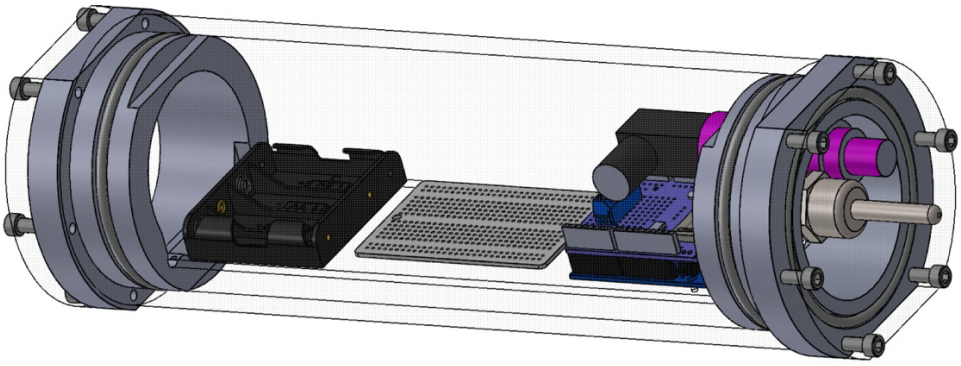
PART 1.2: START BUILDING THE INSTRUMENT

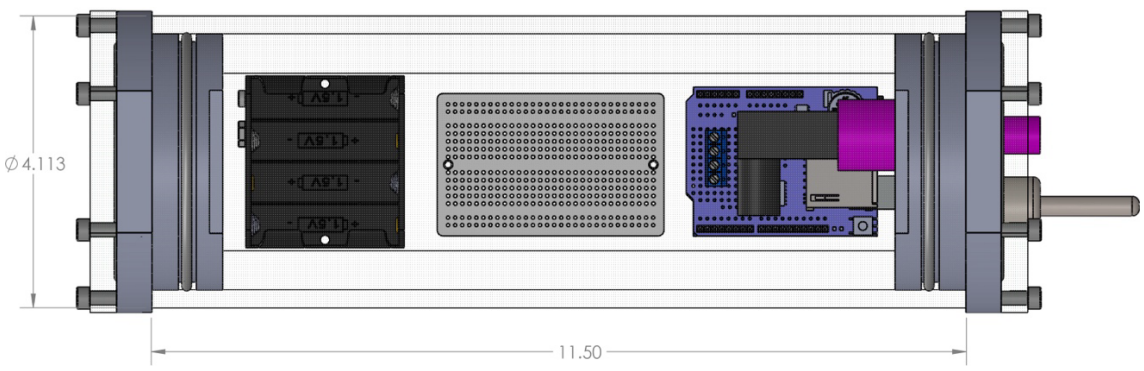
There are several steps you'll need to take to deploy your sensor in the field: you will need it to run without the computer and protect it from the elements. You'll need power for the Arduino, have a way to record the data, and create a waterproof housing.

STEP 1: TRANSFER YOUR CIRCUIT FROM THE BREAD BOARD TO A PERF BOARD

Start with your circuit with the Adafruit thermistor set up on your breadboard and working. You'll need to transfer your prototype wiring from the breadboard to more permanent wiring to deploy in the field. You may have noticed that wires in breadboards can be easily jostled, and your field instrument needs to be jostle-proofed. You will use a **prototyping board (AKA perf board)** and will solder the wires that you are currently connecting with the breadboard. Soldering is much more permanent than a breadboard, and although you can unsolder things if you make a mistake (which you will!), it can be very difficult to unsolder some components. It is very important that you lay out your components on the perf board logically, so before we do this, let's take a step back and think about the whole project.

Ultimately, you are going to have a temperature sensor (the Adafruit thermistor) and a pressure sensor that send data to the Arduino. All of the electronics need to be in a waterproof housing, and the temperature and pressure sensors need to be able to "sense" outside of the waterproof housing, somewhat like the figures below:





Use the figure to orient your perf board and Arduino – the idea is to figure out how many wires connect different parts and think about how the wires and parts should be oriented.

A few things to think about:

- You'll want to use screw terminals to make the electrical connections from perf board to the Arduino and the sensors to the perf board so you can connect and disconnect them easily. How many screw terminals do you need for the thermistor set-up? How should the screw terminals be oriented so you can easily insert and remove wires from them?
- The perf board has connected rows on both sides for power and ground. The pressure sensor that you will set up in Part 2 requires 12V, but the thermistor and Arduino need 5V. So you'll want to establish (and label!) one side as 5V and one side as 12V.
- We will use this same perf board for the pressure sensor project next week, so set up your wiring so you leave space on the board for that circuit.
- Lay out your circuit to avoid having too many crossed wires or risk connecting wires that should not be connected (how much do you trust your soldering ability?).
- In a circuit, you generally (with some exceptions) want all ground wires to be connected to each other – this is called establishing a common ground.

Before you start soldering, run your plan by Grant or Kelly.

Once you've gotten your circuit transferred to the perf board, check to make sure your sensor is still functioning with your Arduino.

STEP 2: NEXT MOUNT YOUR PARTS ON ACRYLIC.

Attach the SD card shield to the Arduino using the headers and then mount the perf board and the Arduino to a piece of acrylic to fit inside the waterproof housing.

A) Get a piece of acrylic (~2.5" wide x ~10" long) – this will be pre-cut. Using the diagram above as a guild, mark the location of the mounting holes on your Arduino and perf board on your piece of acrylic using a transfer punch. Leave room at one end for a battery pack. (Check with Grant or Kelly before drilling!) You may, or may not need to use all of the mounting holes! If necessary, you can en-bigen (make bigger, enlarge! That's the word!) the mark left by your transfer punch with an auto-punch (Ask for assistance if you are unsure how to use the tools!).

B) You will then use a #43 drill bit (0.089in diameter) to make holes at the locations you have marked on your acrylic. It is important to stay as square (perpendicular in 2 axes, i.e., DRILL STRAIGHT!) as possible when drilling the holes (using a drill press or milling machine makes this not a problem!) to make your life easier later on.

C) Once the holes are drilled, use a 4-40 (#4 diameter, 40 threads per inch TPI) tap to cut threads in the holes you just drilled. NOTE: THIS IS A SMALL TAP AND IT IS VERY FRAGILE!!! If this is new for you, please ask for help!

D) If your acrylic has paper on it, take it off before mounting the boards. When all of your holes are tapped 4-40, install 4-40 standoffs in the perf board holes and use 4-40 Phillips head screws to attach the perf board to your acrylic.

E) Repeat this for the Arduino (NOTE: You may be able to screw your Arduino directly to the acrylic w/out standoffs… if this is the case, simply use 4-40 screws).

PART 1.3: LOGGING YOUR DATA

Once you have your hardware mounted, it's time to set up the **SD card shield** to record your data onto an SD card. Here's the site for the SD card shield:
https://learn.adafruit.com/adafruit-data-logger-shield

STEP 1: SET UP THE REAL TIME CLOCK (RTC)

You should have already soldered the headers onto the board, so move on in the tutorial for setting up the SD card shield to log data. For logging data, it's very handy to have an accurate time reading with each data point so you can plot your data and compare it to data from other instruments that were collected in the same time frame. Conveniently (and because this is such a common need), the SD card shield has a real time clock (RTC) chip built in.

Set up the RTC (real time clock), following the directions on the adafruit site.
https://learn.adafruit.com/adafruit-data-logger-shield/using-the-real-time-clock

You'll need to run a calibration code once to set up the time on the SD card shield and then as long as you leave the battery in, the time should stay accurate (that said, you should check it before deploying!). The real time clock (RTC) chip you have on the SD card shield should be labelled on the shield – look for "DC1307" or "PCF8523" on the card shield and then find the right example in the RTCLib folder of examples [File -> Examples -> RTCLib]. If you don't see "RTCLib" under "Examples," you will need to add the RTCLib library [Tools -> Manage Libraries]. You should run the example code once and then check when you set up the SD card logging today to make sure it is reading the right date and time. [Note: Check your time carefully and make sure it matches the time on your computer – don't "verify" the code before uploading – this can cause the time to be off.]

STEP 2: WRITE THE SOFTWARE TO LOG DATA TO THE SD CARD

We are going to start with an example code from the Adafruit website that was written as part of a project to log temperature and light. Note that this project is NOT the same as our project, so we are going to need to read the code carefully so we understand what it is doing and which parts we should keep and which parts we should delete. The other option would be to write a new code ourselves, but it is usually faster to find existing code (there is A LOT on the internet!) and modify it, while being cautious about putting too much trust in code that we downloaded from some website.

Download the adafruit example code here:
https://learn.adafruit.com/adafruit-data-logger-shield/using-the-real-time-clock-3

You want to use the SD card shield to log temperature that you measure using your thermistor circuit. To do this, you'll have to combine two programs – the example for the data logger shield that you just downloaded and the program you are using to measure temperature. You should record your data in a csv file.

You should download the complete code from github to make sure you have all the parts of the code. Read through the code first. It's a good idea to upload the program to check to make sure it's working (i.e., adding files to your SD card), before you start making changes. You will have to make some changes to the code.

- The example is logging light on A0 and temperature on A1. We are logging temperature on A0 and (will be) logging pressure on A1. Rename the variables so you don't get confused – e.g., "photocellPin" should be "thermPin" and "tempPin" should be "pressurePin." I recommend using find and replace so you don't miss any examples.
- We are not going to use the variables "BANDGAPREF", "aref_voltage" or "bandgap_voltage" so go ahead and delete those variables as well as the sections of code that use them.
- The example code has RTC_DS1307 – we are using an upgraded SD card shield. Your SD card should say what RTC you have (in small print near the battery), e.g., PCF8523. Change the text in your code to define the Real Time Clock object.
- Change the text headings in the void setup loop to what your instrument is measuring. You might need to come back and adjust this once you get farther in the code.
- Note that the code includes calculations for temperature – DO NOT USE THESE! Delete and insert the calculations from your temperature code. Note that instead of defining the variables at the beginning, they defined variables that are not constant (such as "voltage" and "temperatureC") by typing "float" before the new variable name. Make sure you either define the variables at the beginning or in the line that you first introduce them. Take the time to keep your variables straight when doing this and keep track of your units. If your temperature code is working well, you should be able to copy it over.
- You can adjust what data are recorded on the SD card – to get the SD card logging, it's fine to just log the Arduino reading from A0 but you will eventually need to decide which variables you will want to log on the SD card. Make sure that in your code, you do the same thing with "logfile.print" and "Serial.print" – this is important for troubleshooting.

You currently have a lot of different variables printed to your serial monitor. Consider which variables do you think you need to record on the SD card? What factors should you consider in making this decision? For now, you should set up your SD card to log millis, date and time, and some measure of temperature. You will add the rest of the calculations to the code later.

Check to make sure your code is working by re-uploading your Arduino sketch to log data a few times – you should end up a different file on your SD card for each time you started and stopped logging and the data should match what you see in the serial monitor. Note that if you have a new SD card, you might need to format it before using it.

HOMEWORK

15. Complete the MATLAB tutorial and include the certificate in your lab report (15 pts)
16. Log temperature in your room or in the lab **for at least 24 hours** and save your data to your SD card. For this exercise, you will power your instrument with the power adapter. Graph your temperature data as a function of time and turn in the graph with your lab report. Use MATLAB to generate your plot. I recommend that you try to plot real date and time rather than millis but if you are new to MATLAB, it is fine to plot mills for this part (+2 bonus points for plotting real date and time). See instructions for Part 4.1 – data analysis. (30 pts)

Some troubleshooting tips for Arduino:

The best way to combine two programs is to start by stripping down any unneeded lines of code from both programs. This includes commented out lines or sections. Next you should make sure that everything is annotated very well. Then starting with the more complex program as a template, begin to add in chunks of the 2nd program in sections. Start with pre-program section (the part before VoidSetup), and add in any libraries and variables. I like to try to keep them separated by original program by adding some blank lines or a bunch of characters (i.e. //%%%).
Then move on to the VoidSetup and copy/paste in necessary lines, and not repeating anything (i.e. if you already have a Serial.begin(9600); line, then you don't need to add it again). Repeat again for VoidLoop.

- ANNOTATE!
- It's a good idea to keep a version of the downloaded code to compare with your code that you are making changes in. If you're not sure whether you should delete some code, try commenting it out first so you can undo your changes (highlight the lines then press "command /", at least on a mac).
- Print outputs or just words to the serial output – more information can help you pinpoint where a problem might be.
- If part of the code isn't working, you can try to comment out other sections and uncomment them sequentially to try to find the problem. In this code, there are sections associated with the RTC, with logging on the SD card, and with reading the analog input and manipulating data.
- If the SD card doesn't have any data on it, check to see if there's a red flashing light on the card shield (says SPI) – this should be flashing at the frequency that you are syncing data (set in SYNC_INTERVAL). Compare the last ~8 lines of code to the original code you downloaded.
- If the SD card has different data than your serial monitor, check to make sure you are including lines of code for both "logfile.print" and "Serial.print" for everything you print.
- If you get stuck, take a break and look at your code later with fresh eyes. If you're still stuck, ask for help!

[Note: There are no questions to answer for this section but you need to complete it to be able to deploy your instrument!]

If you are done early, jump to Part 3.1 and finish the software.

IN CLASS

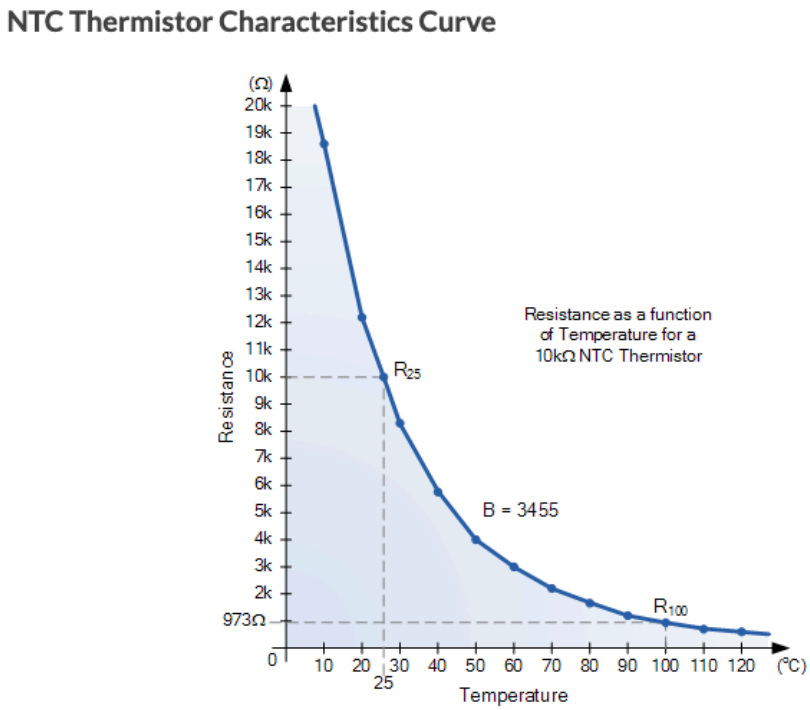
Add this calculation in your Arduino sketch – you'll need to set up new variables at the beginning of the sketch, then add the equation after you calculate your output resistance variable. It's a good idea to make one change to your code at a time and then check to make sure it's doing what you think it should be doing, so set up your thermistor BEFORE you start coding. Annotate your code with descriptions of what you are doing and the units of each variable.

Be careful to use floats for variables that have decimals – if your output is a constant or isn't changing, check to make sure your floats haven't been converted to ints
https://www.arduino.cc/reference/en/language/variables/data-types/float/

Adafruit has more example code to do the calculations here, but note that they wired their circuit differently than we did (in the pull-down configuration), so you should not copy their code directly:
https://learn.adafruit.com/thermistor/using-a-thermistor

It may be useful to read through their code to figure out how to write yours, e.g., if you read their code, you can figure out how to make Arduino code calculate a natural log. They do tend to use a lot of shorthand in their code that makes the code a little more concise but can be a bit challenging for beginners to follow.

Note that this is a $4 part and some code someone you don't know put on the internet, so it's a good idea to check it and do your own calibration if you need a high level of accuracy or if your checks are fairly far off. Check it by comparing the room temperature reading to a measured temperature reading and then submerging the sensor in ice water and measuring the reading. If these calculations are not accurate, you can calibrate the sensor yourself. This is a bit tricky as the relationship between resistance and temperature is nonlinear – see https://www.electronics-tutorials.ws/io/thermistors.html



Since this relationship is nonlinear, you should either work within a narrow band (that can be approximated as linear), or better yet, use the format from the Steinhart-Hart equation and take several measurements and plot (1/T) on the y-axis and ln(R) on the x-axis and fit a linear relationship. Given the time constraints of class, we will opt for a 3rd option, to use the equation given and then post-calibrate our data against a known sensor after we deploy our sensors (time permitting). If you're interested in learning more about calibration, please ask!

Part 2: Pressure Sensor

PROGRESS ON SENSOR SO FAR

At this point, you should have:

1. Set up a voltage divider circuit to measure temperature using a thermistor and Arduino
2. Modified an Arduino sketch to read the temperature sensors
3. Learned about other types of sensors (potentiometers and digital temp sensors)
4. Set up the hardware part of the SD card shield to add to the Arduino to log your data
5. Soldered your circuit onto a perf board that connects to the SD card to measure temperature
6. Mounted the perf board and Arduino on acrylic
7. Set up the SD card logger
8. Collected temperature data!
9. Graphed data!

The next steps are:

1. Set up the circuit to use a pressure sensor to measure depth and get readings in Volts (in a new Arduino sketch)
2. Add the pressure sensor circuit to the perf board
3. Calibrate the pressure sensor
4. Determine the size of battery we need to run our instrument
5. Combine your Arduino sketches to obtain one sketch that reads temperature and pressure and logs the data to the SD card
6. Untether your instrument (i.e., connect the battery), put the instrument in a waterproof housing, and test it
7. Collect data underwater!

OVERVIEW

Part 2.1: Learn about pressure sensors

- Background question to answer before class
- Design and connect electrical components to measure pressure with a 4-20mA output pressure sensor
- Write software to measure pressure using an Arduino microcontroller

Part 2.2: Continue building the instrument

- Add pressure sensor to perf board with temp sensor

Part 2.3: Software and calibration

- Calibrate the pressure sensor
- Integrate the pressure sensor code into software for the temperature sensor (time permitting)

PART 2.1: LEARN ABOUT PRESSURE SENSORS

BEFORE CLASS

BACKGROUND READING

Please read through this information before class. Don't worry about understanding all of it – you will likely need to refer back to these sites during class.

Different sensor outputs:
https://www.apgsensors.com/about-us/blog/taking-the-mystery-out-of-sensor-outputs
https://www.geosense.co.uk/technical/sensor-outputs

Before class, review the reading on sensors and transducers:
https://www.electronics-tutorials.ws/io/io_1.html

GET READY TO SET UP PRESSURE SENSORS (ANSWER #1-7 BEFORE CLASS)

NOTE: These questions will be checked in class for completion (as part of your participation grade), not accuracy. They will be graded for accuracy in the lab report that you turn in.

Your pressure sensor is a gage sensor, which means it measures pressure above a baseline atmospheric pressure. As you know, barometric pressure varies some over time. We want to make sure this variability is small enough that we can ignore it when we deploy our instrument. If a weather system comes through and changes barometric pressure, will that affect our water depth measurements? To determine this, we are going to start by comparing a ballpark estimate of the pressure change from weather-related changes in barometric pressure to the resolution of our sensor. You can get barometric pressure from the Dauphin Island ARCOS station, https://arcos.disl.org/ .

1. Look at the last week of data – what is the range of variability (max barometric pressure – minimum barometric pressure)? Make sure you keep track of your units! (4 pts)

The pressure sensors we are using are 0-0.2 MPa, which is equivalent to 0-29 psi. We will be using both metric (Pa) and imperial (psi) units, so be familiar with the conversions (or just google it). Remember the equation for hydrostatic pressure,

$$P = \rho g h$$

P is pressure (Pa), ρ is density (kg m^{-3}), g is the gravitational constant (9.8 m s^{-2}), and h is height (m)

2. a) Note the units on the terms in this equation. Show that the units on each side are equal. (3 pts)

   b) What is the maximum water depth that we can measure with our 0.2 MPa pressure sensors? (3 pts)

3. What depth resolution will we have with this pressure sensor and the 10-bit resolution of the Arduino analog input? (Hint: consider the maximum water depth and the number of increments of resolution the Arduino has. If you're not totally clear on what "resolution" is, here's a good resource: https://www.youtube.com/watch?v=UlFH6HdoN9k) (5 pts)

Until now, we have only talked about Voltage, and not really mentioned Current. Our pressure sensors are technically pressure transducers that output current, specifically in the range of 4-20 mA. Transducers are devices that convert an input quantity (measured by a sensor) to a different type of output, in this case, to a current output that is directly proportional to the pressure. See here for an explanation:
https://www.omega.com/en-us/resources/pressure-transducers-vs-pressure-sensors

Before we start using the pressure transducers, let's learn a little more about current. Again, the internet is a wealth of information – here's a good tutorial on current and voltage with videos that explain voltage, current, and Ohm's law (~5 min each):
https://learn.sparkfun.com/tutorials/voltage-current-resistance-and-ohms-law/all

4. So in your own words, tell me: what is voltage? And what is current? (8 pts)

Since Arduinos don't read current, we will need to set up a circuit to output voltage (sort of like we did with the variable resistor). The pressure sensor has 2 wires, red (+) and black (-). Convention says that the red wire (+) should be on the high side of the connection, and black (-) on the low side. Because our pressure reading device is a TRANSDUCER, not a SENSOR, it has internal circuits that convert the pressure at the pressure port into a CURRENT that ranges from 4-20mA. This is a common output for many kinds of transducers and has some surprising advantages over a voltage output!

5. Based on your background reading, what are some advantages and disadvantages of 4-20 mA sensor outputs? (6 pts)

The voltage requirement for this sensor (red wire) is at least 12V. The output (black wire) is a current that varies between 4 and 20 mA. You may be wondering why the range is 4-20 mA rather than 0-20 mA – if it were 0-20 mA, you wouldn't be able to distinguish a 0 reading from a disconnect in the circuit. Setting the 0 limit a little higher makes it much easier to trouble-shoot problems with your circuit. Unfortunately, the Arduino can't read current, but recall that Arduinos can't read resistance either, but with some math and smart circuit design we were able to convert resistance to voltage. We can do the same thing with current! How do we convert a current into a voltage? If we define a few boxes, it is easy using Ohm's Law.

Ohm's law states:

$$V = IR$$

Where V is voltage (V), I is current (Amps), and R is resistance (Ω).

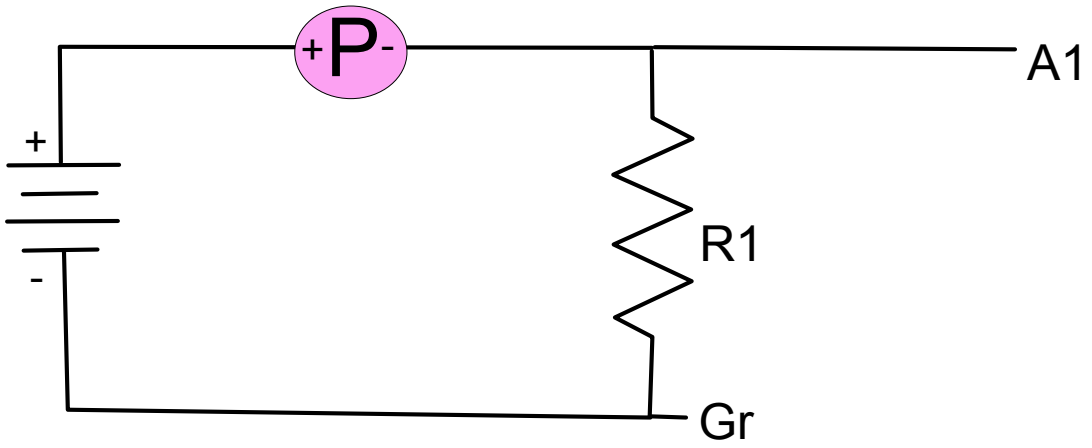


Figure 1: The schematic diagram shows the pressure sensor (P) with the + (red) wire hooked up to the power source and the – (black) wire as the signal (A1). The current goes through a resistor (R1) before connecting to ground (Gr).

The first thing to realize is that the pressure transducer requires a minimum of 12vdc to function correctly. Show on the diagram where the 12v needs to be located.

That 12v is handled by the transducer, which converts it into 4-20mA. The pressure sensors are 0-0.2 MPa, which is ~0-29 psi, and are gage sensors, which means they measure pressure above a set value that is roughly atmospheric pressure. This means that at the max pressure, 0.2 MPa, the sensor should output 20 mA of current, and at the minimum pressure, 0 MPa, the sensor should output 4 mA of current. We want the Arduino to read the maximum input voltage (5V) when the sensor sends out the maximum current (20 mA, @ 0.2 MPa). Well guess what!!!! Take a look at Ohm's Law (This should be in your head for FOREVER!!!!, MEMORIZE IT!!!).

Again, Ohm's law states:

$$V = IR$$

Where V is voltage (V), I is current (Amps), and R is resistance (Ω). Note that Ohm's law uses Amps and we are measuring milliAmps, and 1000mA = 1A. We are also dealing with resistors that range from Ω to kΩ to MΩ. Keep track of your units when you do these calculations!

When you put a resister between ground and a higher voltage (logic level), there will be a change in either the current that flows OR the voltage ACROSS that resistor. The voltage at A1 is determined by the current coming out of the pressure sensor and the resistor (R1) between ground and the signal. We set the resistor value to get a voltage in the range we want to read for our signal using Ohm's law. We have a current (20mA), we have a desired voltage (5V). All we have to do is solve for the resistance!

6. a) What size resistor (R1) do you need to convert the 4-20mA output to a voltage, given that the Arduino analog in can read a maximum of 5V)? (3 pts)

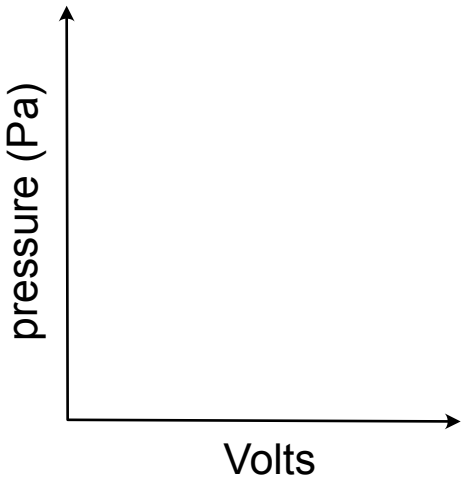
   b) With that resistor, what voltage will the Arduino read when the transducer is supplying 4 mA? (3 pts)

   c) What voltage will the Arduino read when the transducer is supplying 20 mA? (3 pts)

The pressure sensors we are using are cheap knock-offs from China that we bought on amazon, so we have no specs on them. If you buy a high-quality sensor, they often come with a calibration curve equation that you can use to convert your sensor output to the units that you are interested in measuring. Time permitting, we will calibrate our sensors in class, but to get a better intuition for how to do this, we will do the calculations to convert voltage to pressure before class.

We can calculate this by determining the equation for a line that goes through 2 points: the point corresponding to the minimum range of the pressure sensor (y_1) and the minimum voltage (x_1) and the point corresponding to the maximum range of the pressure sensor (y_2) and the maximum voltage output (x_2).

7. Sketch your points on a graph and draw a line through them, then calculate the equation for the line. Give your answer as a mathematical equation and then write it as Arduino code. (8 pts)



[Remember, you can get the slope as

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

Then plug one of the points into the equation $y = mx + b$ to get the intercept (b).]

IN CLASS

PARTS LIST

Most of these parts will be in your toolbox – feel free to use wires and other components as you need them and ask if you're missing anything. This project builds on the temperature sensor project, so you will need your completed set-up with Arduino, SD card shield, perf board, and thermistor mounted on acrylic.

General supplies

- Breadboard
- Multimeter
- Jumper wires for breadboard
- 24gage stranded wire for perf board (red, black, white, yellow)
- Screw terminals
- Resistors

Project-specific supplies:

- 0-0.2 MPa 4-20 mA pressure transducer with tubing connector
- Syringe to connect to pressure sensor
- Long (> 6 ft) piece of tubing to calibrate pressure sensor

ADD PRESSURE SENSOR

Next we will add a **pressure sensor** to the temperature sensor to make a temperature-depth sensor (the TD part of CTD) that we will put in a waterproof housing. With this sensor, we can either measure temporal changes in water height and temperature or collect profile data (like we would with a CTD).

For homework, you should have calculated the size resistor you will need. Look at the resistor options in your kit and find a resistor that is close to the optimal resistor value that you calculated (you may not have the exact resistor size you want). If you are using a different size resistor than you calculated, use Ohm's law again to calculate the voltage output that you expect for the minimum (4 mA) and maximum (20 mA) current from your sensor with the resistor value that you are using in your circuit. If you are using the resistor you wanted, double check the calculations you did for homework before moving on.

The circuit drawn above (Figure 1) is missing something. Recall from the LED project in your starter kit that you needed a resistor in your circuit to light up the LED – why did you need the resistor? What would happen if you made the circuit without the resistor? The answer is that you would have too much current going through the LED and directly to ground. This is a short circuit and will either damage your electronics or prevent them from functioning properly. Similarly, here we do not want unrestricted current going straight into the Arduino (A1) because of the risk of a short circuit that

could damage the Arduino or cause it to malfunction. We are therefore going to add a 10 kΩ current limiting resistor to our circuit (R2) (Figure 2). This is technically a parallel resistor (in contrast to R1, a series resistor) based on its position in the circuit. This is a very deep topic, that Grant is happy to go into detail on, but for this class, it's enough to say that R2 is simply a current limiting resistor going into the Analog pin on the Arduino.

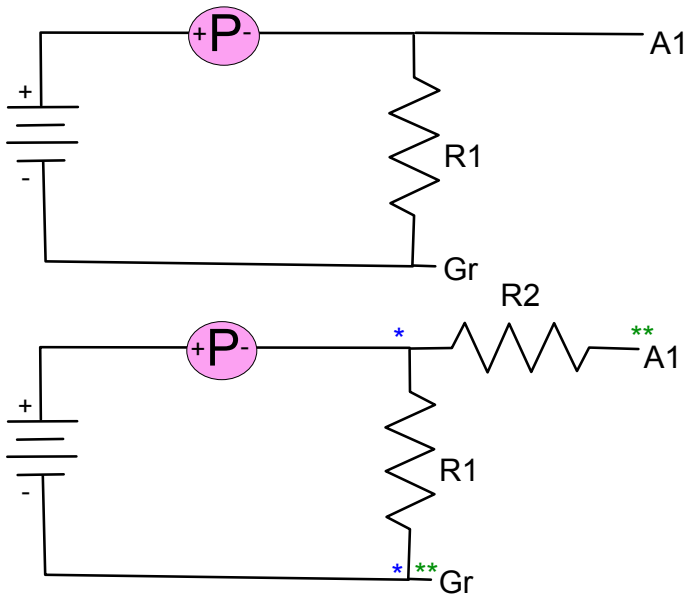


Figure 2: Modified circuit with current limiting resistor (R2). (Top image is Fig. 1 from p. 4.)

Hook up the circuit using your breadboard and the correct resistor (R1). (You'll want to use a screw terminal to connect the wires to the breadboard.) The pressure sensor needs 12V of power, so we will use the AC/DC adapter (wall plug) to power the pressure sensor, so you'll need an adaptor to run the power plug into your breadboard. From now on, we will be using 12V for the pressure sensor and 5V for the thermistor – be VERY CAREFUL about which power source you are using! Before you hook your circuit up to the Arduino, use a multimeter to measure the current and voltage coming out of the pressure sensor (between the * in the diagram above). Attach a tubing adaptor and syringe to the pressure sensor and increase the pressure on your sensor and see how this affects the voltage and current.

8. a) What is the current and voltage between the (*) in the diagram above when the pressure is close to 0? (4 pts)

   b) BEFORE YOU MEASURE, how do you think the current and voltage between (**) will differ from the current and voltage between (*)? (2 pts)

   c) Now measure with your multimeter - what is the current and voltage between (**)? (4 pts)

   d) Was this what you predicted? Why or why not? What does R_2 do in your circuit? (8 pts)

e) OPTIONAL Calculate the current between (**), then check your calculations with your multimeter. For help doing this, see https://www.electronics-tutorials.ws/dccircuits/current-divider.html (3 bonus points)

Open the Arduino program. Remember what you did to measure the analog voltage from the temperature sensor. Following that procedure, get a new sketch running that outputs raw pressure readings in the serial monitor. Save this as a new sketch (maybe call it ReadPressure). Note that we are using two separate Arduino sketches for temperature and pressure, but we will combine them later.

Next modify your sketch to convert the raw pressure readings to voltage (in a new variable), as you did for the thermistor readings. Check to make sure your readings match your calculations from #7 above.

Use the equation from your homework that relates the pressure to volts (0-5V) to convert the voltage from the analog input pin into a pressure and have the Arduino print this (and the other values you have found) to the serial monitor. What should you be getting for a value with the pressure transducer sitting open to the air on the bench?

PART 2.2: CONTINUE BUILDING THE INSTRUMENT

Now that we know the circuit is working, the next step is to **add it to the perf board**. Use the same guidelines that you used when designing your perf board for the thermistor to figure out how to lay out your pressure sensor circuit on the perf board. Check with Grant or Kelly before soldering. You'll have to take the perf board off the acrylic to do this and then re-attach it when you're done.

PART 2.3: SOFTWARE AND CALIBRATION

Now that the board is mounted securely, we are going to check the **pressure sensor calibration.** We can calibrate the pressure sensor using a tube filled with water to different heights above the sensor and use the hydrostatic equation to calculate pressure from water height.

$$P = \rho g h$$

P is pressure (Pa), ρ is density (kg m^{-3}), g is the gravitational constant (9.8 m s^{-2}), and h is height (m)

9. Ideally we would calibrate for pressures over the entire pressure range of the sensor – how long of a tube of water would we need to do this? (3 pts)

Since the water height we would need is a bit challenging to manage in the lab (note the hint on the above question!) and doing a proper calibration curve is time-consuming, we are going to tentatively trust our calculations but check a couple of values to make sure we're not too far off. First with your pressure sensor open to the atmosphere, check your calculated pressure.

10. What do you expect your pressure to be in Pa AND in raw Arduino output units? What pressure measurement do you get in each of these units? (8 pts)

11. What is the precision of your measurement in raw Arduino output units and in Pa? How comfortable are you with your reading based on the precision of your measurement? (6 pts)

Next fill a long piece of tubing (at least 6 ft) with water by using a syringe to suck water from a beaker of water through the tubing. Keep the syringe attached and insert the water-filled end into the pressure sensor fitting. With a partner, hold the tubing with the syringe above the pressure sensor, then remove the syringe. Hold the tubing so the top of the water is 1 m above the pressure sensor (any measured distance will work but aim for at least 1 m) and note the pressure.

12. a) How much does the pressure increase with 1 m of water head on the sensor (again, in Pa and in raw Arduino output units)? (3 pts)

    b) How much do you expect the pressure to increase based on the hydrostatic equation? (3 pts)

    c) Are these values close enough given the precision of your instrument? (3 pts)

OPTIONAL (IN CLASS)

If you have time, record the Arduino output (0-1023) for at least 4 different pressure head heights. Convert Arduino output to volts and height to pressure (Pa), then plot pressure (calculated from the hydrostatic equation) as a function of volts. Fit a line and then compare the fitted line from your calibration to the equation that you calculated based on the range of the pressure sensor. [Note: these calculations are not hard if you're familiar with MATLAB but will take some time if you're a beginner. It is theoretically possible to do this in excel, but graphing in excel is very painful if you are familiar with MATLAB! If you want to do this part, please ask for help, but know that Kelly does NOT graph in excel.] (3 bonus points)

IN OR AFTER CLASS

13. a) What is the depth precision of your instrument (in m or cm)? (2 pts)

    b) If you are recording depth in meters, how many decimal places should you record? (Or in other words, if your program converts the Arduino reading to 3.24163 m depth, how much do you trust that number? What number should you record based on what you know about the precision of your instrument?) (3 pts)

14. Go back to your ballpark estimate of variability in barometric pressure. What is the resolution of your pressure measurements, or how much does pressure need to change for you to detect it at the resolution that you are reading your sensor? Do you need to consider the effect of barometric pressure changes in measuring depth with your instrument? (5 pts)

Part 3: Preparing for Field Deployment

OVERVIEW

Part 3.1: Finish the software
- Plan coding for temperature calculation before class
- Calculate temperature in Arduino code
- Calibrate the sensor / check temperature calculation

Part 3.2: Untethering and powering your instrument
- Determine current draw of instrument to do battery calculations
- Learn about batteries and calculate battery requirements
- Add a battery pack to your instrument

Part 3.3: Waterproof your instrument
- Put instruments in waterproof housing
- Go through checklist – double check so no flooding!
- Deploy off dock and collect data!

PART 3.1: FINISH THE SOFTWARE

By the end of this section, you should have one code that measures and converts temperature, measures and converts pressure, and logs the data to your SD card at a time interval that you specify. Right now, you should have two separate codes – one that measures temperature (but not yet in °C) and logs to the SD card and one that measures pressure. When merging two codes, it's a good idea to start with the longer or more complicated one (in our case, the temperature logging one), and copy and paste the other code into that one in increments, checking to make sure that it's working. It's also a good idea to (1) read through your original code and annotate it, specifically labelling the different sections so it will be easier to integrate the two codes together, and (2) save your Arduino sketch as a new file name before modifying it so you can go back to your old code if the changes you make don't work.

STEP 1: CONVERT RESISTANCE TO °C

Before you incorporate your pressure sensor into your temperature code, you need to convert resistance to temperature in°C.

BEFORE CLASS

Next, you'll need to convert the output from your thermistor from resistance to **°C**. You can do this either through calculations based on theory or by **calibrating the sensor.** We can use the Steinhart-Hart equation:
https://en.wikipedia.org/wiki/Steinhart%E2%80%93Hart_equation
to convert the resistance of our thermistor to temperature.

$$\frac{1}{T} = \frac{1}{T_{25}} + \frac{1}{B}\ln\left(\frac{R}{R_{25}}\right)$$

T_{25} = room temperature (298.15K)
B = coefficient of thermistor (3950… This is a number you look up on the spec sheet!)
R_{25} = resistance of thermistor at 25C (= 298.15K) (10kΩ)
R = resistance measured (current resistance at current temperature)
T = temperature in Kelvin

Look at your Arduino sketch and think about how you will do this calculation in your code. We are going to actually write the code in class where you can check each change you make with your instrument, but you should come to class with a plan. What variables will you need to create? It's a good idea, especially when you are learning to code, to separate calculations into different lines of code, so here you probably want to set up a variable for 1/T (call it, say inverseTemp),

then in the next line calculate T. You can do all of your calculations in one line of code if you are feeling confident, but if you have an error, it's easier to trouble-shoot if you have multiple lines of code with a step on each line. Write ON PAPER the lines of code that you think you will need to add to your script.

1. Include your notes on writing your code in your lab report (a photo is fine!) (3 pts).

IN CLASS

Add this calculation in your Arduino sketch – you'll need to set up new variables at the beginning of the sketch, then add the equation after you calculate your output resistance variable. It's a good idea to make one change to your code at a time and then check to make sure it's doing what you think it should be doing, so set up your thermistor BEFORE you start coding. Annotate your code with descriptions of what you are doing and the units of each variable.

Be careful to use floats for variables that have decimals – if your output is a constant or isn't changing, check to make sure your floats haven't been converted to ints
https://www.arduino.cc/reference/en/language/variables/data-types/float/

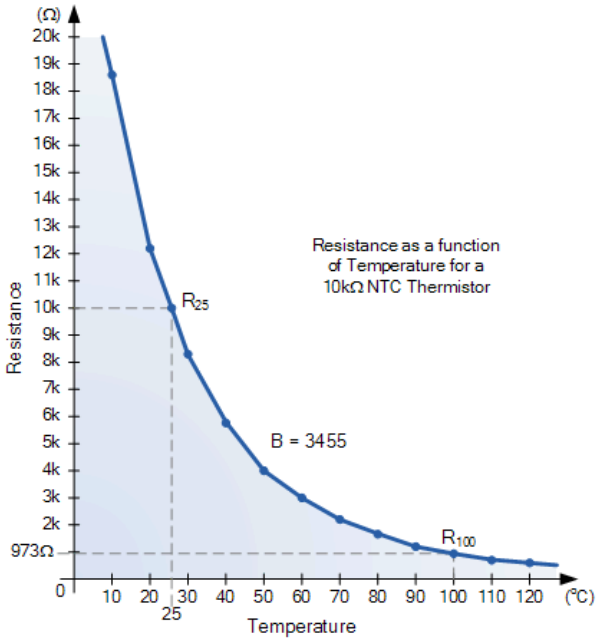
Adafruit has more example code to do the calculations here, but note that they wired their circuit differently than we did (in the pull-down configuration), so you should not copy their code directly:
https://learn.adafruit.com/thermistor/using-a-thermistor

It may be useful to read through their code to figure out how to write yours, e.g., if you read their code, you can figure out how to calculate a natural log in Arduino code. They do tend to use a lot of shorthand in their code that makes the code a little more concise but can be a bit challenging for beginners to follow.

Note that this is a $4 part and some code someone you don't know put on the internet, so it's a good idea to check it and do your own calibration if you need a high level of accuracy or if your checks are fairly far off. Check it by comparing the room temperature reading to a measured temperature reading and then submerging the sensor in ice water and measuring the reading. If these calculations are not accurate, you can calibrate the sensor yourself. This is a bit tricky as the relationship between resistance and temperature is nonlinear – see
https://www.electronics-tutorials.ws/io/thermistors.html

Since this relationship is nonlinear, you should either work within a narrow band (that can be approximated as linear), or better yet, use the format from the Steinhart-Hart equation and take several measurements and plot (1/T) on the y-axis and ln(R) on the x-axis and fit a linear relationship. Given the time constraints of class, we will opt for a 3rd option, to use the equation given and then post-calibrate our data against a known sensor after we deploy our sensors (time permitting). If you're interested in learning more about calibration, please ask!

NTC Thermistor Characteristics Curve



STEP 2: ADD YOUR PRESSURE MEASUREMENT CODE

Next, you'll need to add your pressure measurement code to your code that measures and converts temperature and logs it to the SD card. I recommend doing this while the instrument is hooked up to your computer so you can copy and paste segments of code a few lines at a time and make sure the code is still working.

PART 3.2 UNTETHERING AND POWERING YOUR INSTRUMENT

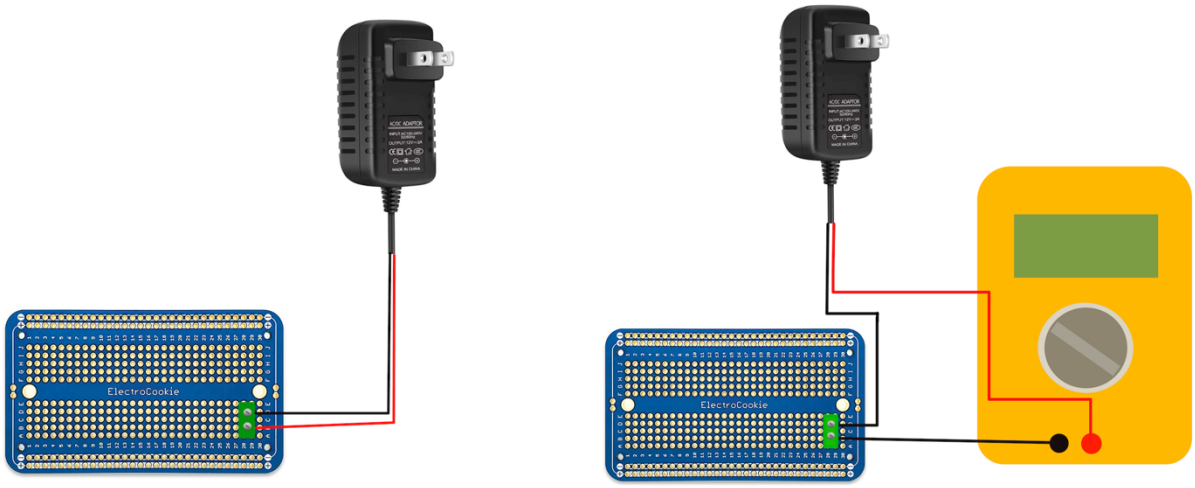
IN CLASS

These instruments are going to go to the bottom of the ocean (or Mobile Bay), thus need to be fully self-contained, not tethered to your laptop or a power source. We are going to do this in two steps: (1) unplug the instrument from your laptop, and power it using the AC to DC power adapter (that plugs into a power outlet), and (2) replace the power adapter with a battery pack that fits in the underwater housing.

2. What components of your instrument require power and how much power (in V) does each require? We need to have one power source for all components of our instrument – how do we achieve this? (Hint: think about or look up how the Arduino handles power) (6 pts)

Come up with a plan to power your instrument from the AC to DC power adapter – run it by Grant or Kelly before soldering if you're not sure about your plan!

In order to **determine what size battery we need** to use for our instrument (or how long a battery pack will run the instrument), we first need to know how much voltage and how much current the instrument needs. We are going to measure current by hooking up the multimeter such that the current goes **through** the multimeter into the circuit (see Figure). This means that instead of power and ground wires connecting from the wall plug directly to your instrument, the power wire connects to the multimeter (red probe), the current travels through the multimeter and out through the black probe, which is connected to the power wire of your instrument. You can use pigtail power adapter connectors (pictured on right) to put your multimeter in the line of your circuit.



3. What is the current draw of the instrument? (4 pts)

Write this number down – you'll need it for your homework (below). Keep track of your units – the next step requires Amps, not mA!

Note that this number will vary depending on how often you are writing data and various other factors, so consider it a very rough estimate and plan your battery needs very conservatively.

AFTER CLASS

BATTERY CALCULATIONS

To run your instrument on a **battery**, which you'll need to do in the field, you will need to develop/construct a power supply for your datalogger (it's more than just an Arduino now!!). To do this, you will need to identify, and understand, some of the technical specifications (specs) of your logger. What voltage does the Arduino use (engineers call this "logic level")? NOTE: This is different from the SUPPLY voltage (the voltage of a power supply used to power the Arduino). To make sure the Arduino always has the exact amount of voltage, the SUPPLY voltage is actually higher than the voltage the Arduino needs, and the supply voltage goes through a voltage regulator on the Arduino that reduces the voltage to a constant value. Why is it important that the Arduino has a constant voltage?

4. What voltage should our constructed power supply be to run the Arduino and the pressure sensor? (4 pts)

Once you have measured your current draw for your instrument, you can use the formula below to determine the MINIMUM capacity (C) of your power supply for the amount of time you want to run your system (or alternately, the amount of time you can run your instrument on a given battery capacity).

$$C = xT$$

C = capacity in amp hours
x = current draw in amps
T = time in hours

How to calculate battery run-time
https://www.powerstream.com/battery-capacity-calculations.htm

For example, if your datalogger is drawing 120mA and you want to run it for 12h:

C=0.12A * 12h = 1.44Ah

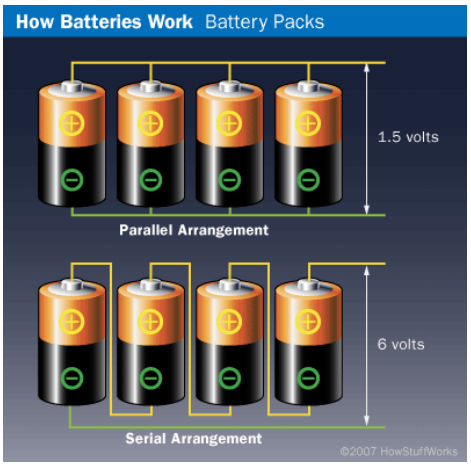
5. Calculate the minimum required capacity of your power supply to run your datalogger for 24 hours AND for 1 week. (6 pts)

For our first deployment, we are testing our instrument with a battery pack with 8 AA batteries.

6. What is the nominal voltage of a AA battery? What is the capacity (Amp hours) for those batteries? (google this, check a couple websites – do they say the same thing?) (4 pts)

Batteries can be arranged either in series or in parallel (see Figure). If arranged in series, the voltage is combined (1.5V x (number of batteries in series)). If in parallel, the capacity is combined (for a AAA… 1Ah x (number of batteries in parallel)).

7. a) What arrangement are the batteries in our battery pack? (3 pts)



b) What is the output voltage and output capacity of the battery pack? (4 pts)

c) How many hours do you predict your instrument will run on this battery pack? (4 pts)

(NOTE: It's always good to oversize the capacity of your power supply if you have the space… too much current isn't a problem (Ah), too much voltage IS)

d) If we used 8 D batteries instead of 8 AA batteries, how long would our instrument run? (2 pts)

PART 3.3: WATERPROOF HOUSING

IN CLASS

Next you'll need to get your waterproof housing ready.

Waterproof housings are difficult and time-consuming to construct. This is why underwater instrumentation is so expensive. This is something that should be considered very early, and lots of time and resources budgeted for the development and testing when planning a project.

The waterproof housings we are using are made from extruded acrylic tubing and acetal (Delrin) end caps with 3/8-inch cast acrylic covers. The acrylic tubing we are using is fairly expensive because its ID (Internal Diameter) is ground to nominal +/- 0.03in.

8. What does the term "nominal" mean in this context? Explain it in your own words. (2 pts)

While this may seem like a small tolerance, it is quite large when working with static waterproof seals, and as a result, all of the end caps are custom made for each end of each tube. It is important that you don't mix them up, or it is likely that they will not create a complete seal and leak, destroying your electronics! Each end cap is labeled with a letter (i.e. "A") and either a "1" or a "2". One end of the pipe has the letter and a "1" and should be mated with "A1" end cap. The unlabeled end of the pipe should be mated with the "A2" end cap.

How deep can these housings go? This is important to know before we throw them in the ocean!

The formula for the External collapsing pressure for a cylinder is

$$p = \frac{t}{R}\left(\frac{\sigma_y}{1 + 4\frac{\sigma_y}{E}\left(\frac{R}{t}\right)^2}\right)$$

Where σ_y is the yield stress (MPa), E is the elastic modulus (MPa), R is the radius of the cylinder, and t is the thickness of the cylinder wall. For acrylic, E = 2550 MPa, σ_y = 68.9 MPa.

9. a) Measure the acrylic tubing to determine R and t (2 pts)

Note: b–e can be answered in or after class

b) Solve the equation for the pressure (3 pts). (Be careful to keep track of your units!)

c) Using the hydrostatic equation, calculate the depth limit of the housing, assuming the limiting factor is external collapsing pressure. (Again, keep track of your units! Double check your work – if you get this wrong and send your housing down too deep, it could implode!) (4 pts)

d) Consider a scenario in which you want to add some additional sensors and battery packs, and your new design requires a housing that is 2X the diameter of our housing. What would be the depth limit of that new housing? (3 pts)

e) What if you want to deploy this new instrument offshore at 100m depth for 3 weeks? What problems might you encounter, and how would you modify the instrument to address these problems? (4 pts)

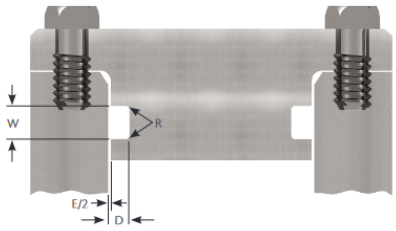
We will be using machined acetal end caps with O-rings to seal the end of our housing. There are two kinds of static seals in the design, radial and axial. The radial seal uses a -237 (dash 237) O-ring and makes the seal between the ID of the tube and the OD of the end cap. Specifications for the fit and feature sizes can be found in Parker O-ring Handbook, which is the industry standard for designing all kinds of O-ring based seals. However, Grant prefers to use Hi-Tech Seals O-ring Design Guide simply because it is slightly less dense, but in either resource, you will find correct information. The Figure below is from Hi-Tech-Seals O-ring Design Guide.

O-Ring Design Guide

Machining Specifications

Groove Dimensions - ISO 3601

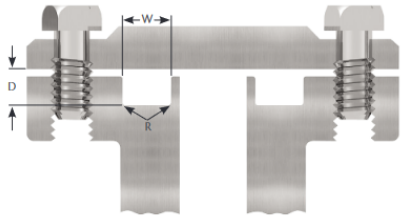
Static Radial Applications



O-Ring C/S	D Groove Depth	Squeeze Inches	Squeeze %	E Diametrical Clearance Max.	W* Groove Width +0.010/-0.000 No Back-Up Ring	W* Groove Width +0.010/-0.000 One Back-Up Ring	W* Groove Width +0.010/-0.000 Two Back-Up Rings	R Groove Radius
0.070	0.049 - 0.057	0.010 - 0.025	14 - 35	0.004	0.110	0.165	0.220	0.008 - 0.016
0.103	0.075 - 0.087	0.013 - 0.031	13 - 30	0.005	0.150	0.205	0.260	0.008 - 0.016
0.139	0.101 - 0.117	0.018 - 0.042	13 - 30	0.006	0.197	0.252	0.307	0.016 - 0.031
0.210	0.156 - 0.180	0.025 - 0.059	12 - 28	0.006	0.283	0.354	0.429	0.016 - 0.031
0.275	0.212 - 0.242	0.028 - 0.069	10 - 25	0.007	0.374	0.484	0.594	0.031 - 0.047

** Pneumatic applications typically do not use a Back-up ring.*

Static Axial (Face) Applications



O-Ring C/S	D Groove Depth +0.004/-0.000	Squeeze %	W Groove Width +0.008/-0.000 Hydraulic	W Groove Width +0.008/-0.000 Pneumatic/Vacuum	R Groove Radius
0.070	0.051	21 - 36	0.126	0.114	0.008 - 0.016
0.103	0.079	19 - 30	0.157	0.142	0.008 - 0.016
0.139	0.106	17 - 26	0.209	0.189	0.016 - 0.031
0.210	0.165	15 - 23	0.299	0.276	0.016 - 0.031
0.275	0.224	13 - 20	0.354	0.335	0.031 - 0.047

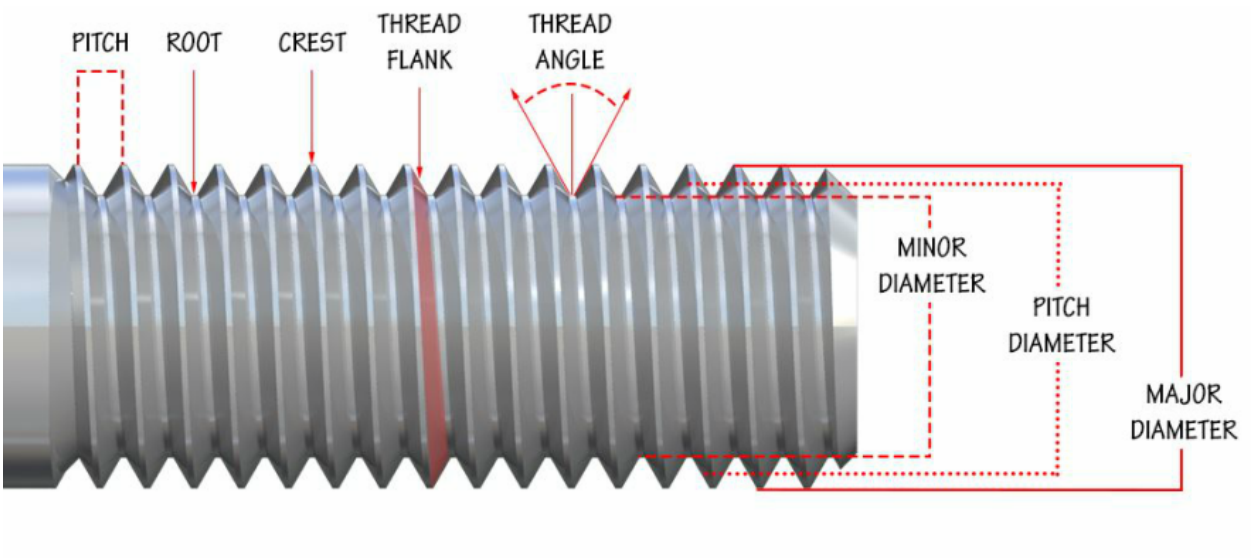
HITECHSEALS.COM 9

The complete guide can be found here if you're interested:
https://www.hitechseals.com/includes/pdf/brochure/O-Ring%20Design%20Guide.pdf

The axial seal on your end cap uses a -235 O-ring and creates the seal between the acrylic cover and the face of the end cap. The acrylic cover is held in place with six (6) 10-32 x 0.75 socket head machine screws. These should only be tightened to about 10 in-lbs (not very much, just past finger tight). The screws are only there to hold the acrylic cover and O-ring in position. The pressure of the water is what makes the seal!

In order for your pressure transducer and thermistor to interact with the water directly, you will need to drill some holes in one of your end caps. You will need to determine what size threads are on the end of your pressure transducer and on your thermowell, then look up what size holes you will need to drill. THE LOCATION OF THE HOLES ARE CRITICAL!!! So measure twice and cut (drill) once.

To determine the thread size, you will need two measurements: Major Diameter and Thread Pitch. Major diameter can be found by using a set of calipers to measure the OD of the threads (not in the root of the threads, but the crest!). You will then use a thread pitch gauge to determine how many TPI (Threads per inch). The thread pitch gauge has a bunch of blades with notches. You should check the threads on your transducer (or thermowell) against the notches on the gauge until you find one that lines up perfectly, then read the number on the blade. This number is the TPI of your thread. With this information, you can look up what size hole you need to drill for 75% thread engagement (also called pitch diameter). See the figure below for examples.



After determining what size threads you will need to cut and marking/laying out where you will drill the holes, speak with Grant or Kelly about how to cut the threads.

Add Teflon tape to your thermowell before installing and be careful not to over-tighten it… doing so will crack the acrylic. Do not add Teflon to the pressure transducer, it has a gasket at the base of the threads that will make the seal (but check to make sure the gasket is there!). Tighten to hand tight only, you should be able to inspect the seal through the acrylic once it's installed.

You should then install the acrylic cover onto the end cap with the six 10-32 screws, capturing the -235 O-ring in its grove. Tighten all six screws until they no longer spin freely using a star pattern, then return to each screw and snug them down until the O-ring seal is complete all the way around the circle.

Once the "sensor end" is installed, install the other end cap without its acrylic cover. Then use the "test acrylic cover" with a syringe to create a vacuum inside your housing. Leave the vacuum in place for a few minutes and watch the gauge, if it changes, then there is a leak, if there are no changes, you should be good to go!

Increase the vacuum inside your housing to 20 in Hg.

10. What depth underwater would this pressure differential represent? Show your calculations. (2 pts)

Once you have leak-tested your housing and are satisfied, you should hook up your datalogger to its power supply so that it is logging pressure and temperature internally. You should then push your temperature probe into the thermowell and hold it in place with hot glue. Connect your pressure transducer to the control board. Your datalogger can then be slid into the housing, and both acrylic covers bolted on, following the same procedure as before. As you are assembling your housing, inspect all parts thoroughly, making sure they are securely in place, and there is no damage. Details matter here, and a tiny piece of sand can flood your entire housing!

This is a good time to verify that your datalogger is still working by making sure you can see flashing LED's.

Next you will need attach a line and weights to your instrument and deploy it off the dock. Ask for specific instructions from Grant or Kelly on where and how you should do this.

Before deploying, go through this checklist:

- Code logging data at a reasonable time interval
- SD card logging data to a new file (do a test run with instrument out of housing)
- SD card in SD card shield (before you plug in the power supply!)
- Ports in waterproof housing tightened completely
- Battery hooked up and light on on Arduino
- O-rings are in place and contacting acrylic (2 per cap)
- Instrument is tied on tightly and weighted sufficiently

After deploying, it's a good idea to check on your device after about an hour or two… Just to make sure it's not filling up with water! Write down the time you check it so you can look at that time in the data once you retrieve it.

11. Based on your estimates of how long your instrument will run, when should you retrieve your instrument? (5 pts)

12. Include a photo of your instrument in its waterproof housing ready to go in the water! (can be you holding the instrument or just the instrument) (35 pts)

Part 4: Field Deployment and Data Analysis

OVERVIEW

Part 4.1: Temperature and depth data from dock deployment
- Import data and plot data in MATLAB
- Compare it to ARCOS weather station data – download, import, and plot data

Part 4.2: Compare data with castable instruments to CTD cast
- Import data and plot in MATLAB
- Plot CTD data in MATLAB
- Interpret data

Part 4.3: Lessons learned and next steps

PART 4.1: TEMPERATURE AND DEPTH DATA FROM DOCK DEPLOYMENT

Let's look at your data! You should have recorded the time in milliseconds, Date/Time from the RTC, temperature (C), and pressure (at least). You can either convert pressure to depth in your Arduino code or once you get the data into MATLAB. We will look at the data in MATLAB.

STEP 1: IMPORT YOUR DATA

The easiest way to load csv data is using the command 'readtable'

```
mydata = readtable("LOGGER01.csv");
```

However, our data includes time formats, which are a bit tricky, so we are going to use the **Import Wizard** in MATLAB. In MATLAB, under "Home" click on "Import data" then click on your csv data file. Using the import wizard, you can adjust how the columns of data are separated (toggle between fixed width and delimited), column names, the name of the output variable, and, importantly, formats of data. Most of your columns are numbers and MATLAB will figure that out, but you will need to specify the datetime format for your datetime variable. Click on that column to highlight it, then click on the format box (it probably says "Text") and scroll down to find the date and time format that you used in recording your arduino data. If you can't find the EXACT match, you might need to define a custom date and time format – hold the cursor over "custom date time format" to get a help box. This is confusing, so ask for help if you get stuck! You might want to rename your output variable something like "mydata" instead of "LOGGER01" but as long as you can keep track of it, you can call it whatever you'd like.

Once you have your data formats right, make sure that the range includes all of your columns and rows of data, select the output format to be "table" and then click on the arrow next to "Import selection" and choose "Generate script" – this will create a MATLAB script that imports your data.

Once your data are imported, you should see a new variable in your MATLAB workspace for your imported data. Double click on it to open it in MATLAB and check to see if it looks right, then check the variable types by using the command *summary*. If you're not sure how to do this, you can google "MATLAB summary" or just type "help summary" in the MATLAB command window. Since this is not part of our code that we will want to run repeatedly, type this command directly into the console and leave off the semicolon so MATLAB will give you your answer.

Dealing with dates and times is a huge pain in MATLAB (and all other programs that I have used). This is one reason why it's useful to also record millis, which are easy to plot quickly to check to make sure your data look okay. To plot with the proper date and time on the x-axis, you need your date and time to be in one column in "datetime" format. Here's a website to help:
https://www.mathworks.com/help/matlab/ref/datetime.html

You can check this with the command *class* and call a specific column in your table using the name of the table followed by a period then the name of the variable (exactly as MATLAB is calling it), e.g.,

```
class(mydata.dateTime)
```

To analyze your data, you can start your script file by running your import code (just type the name of the script file) – this way you will not have to re-import your data each time you change something in your code.

SOME MATLAB TIPS

- It's good practice to type commands in your script and then run the script rather than typing commands directly in the console. So type the command in your script, save the file, and then hit "Run" (green arrow) to run the script.

- I recommend annotating each line of code with the units – in Arduino, you did this with "//" but you can annotate in MATLAB by typing "%" – MATLAB ignores all text in a line that follows a '%"

- Note that in MATLAB, if you don't end a line of code with a semicolon, MATLAB will print out that variable – this is occasionally useful if you have a number or small array and want to see the value right away but generally you want to make sure to use the semicolon to keep your command window from getting cluttered.

- It's a good idea to clear any variables you aren't using and close figure windows. This is an early thing to check when troubleshooting code. You can set this up at the beginning by starting your code with "close all, clear" which will close all figure windows and clear all variables before running the code.

STEP 2: PLOT YOUR DATA

Plot your temperature, pressure, and depth data as a function of time (look up the 'plot' function). You can plot different scales on the same graph using 'yyaxis' or can plot using subplots using 'subplot' or 'tiledlayout.' It's good practice to keep your data in a table and use the variable names within the table "mydata" in which case you need to type "mydata.millis" but you can also create new (shorter) variables, as

```
      t = mydata.timestamp; %date and time in datetime format
temp = mydata.Temperature_C_; % temperature in C
      pressure = …
      depth = …
```

[here you might need an equation to convert pressure to depth if you didn't do this in your Arduino code – keep track of your units!!!!]

To plot, start by simply typing

```
plot(t, temp)
```

Then look up the *plot* command to see how to change the line color and type. You can add another plot on top by typing

```
hold on
plot(t, pressure)
```

However, doing this will plot both temp and pressure on the same scale, which might not be what you want. Try

```
yyaxis left
plot(t, temp)
yyaxis right
plot(t, pressure)
```

That should look better, but you might want to modify the line colors and types. It's a good idea when you are first looking at your data to plot both a line and dots at each sample so you can see whether your sampling frequency was high enough to capture the patterns in your data. You can do this by plotting the same data twice with different input commands, e.g.,

```
plot(t, temp, 'b')
hold on
plot(t, temp, 'r.')
```

This can get messy if you have too many plots, so comment out the line of code plotting all the dots once you check it. Note that MATLAB has a lot of tools to smooth data and remove outliers. We are not going to get into that now, but if you have data points that are clearly outliers, there are ways to deal with them.

Read up on the '*plot*' function in MATLAB and add axis labels to your graph.

1. Save your graph with axis labels and both temperature and depth plotted as a function of time. Which variable(s) are the independent variable(s) and which are the dependent variable(s)? (5 pts)

STEP 3: COMPARE TO WEATHER STATION DATA

To check the functioning of our sensors, we will compare our data to data from the Dauphin Island weather station, which is nearby. Go to arcos.disl.org and download the hydrographic data for the Dauphin Island station for a period slightly longer than the deployment of your sensors. Note that you can view the data on the website by clicking on the station but need to click "Download data" from the left menu to download it to use in MATLAB. [Note: this is supposed to be upgraded at some point – if these instructions don't make sense, that might have happened!]

First load the data. Here there is header information in the csv file, so load it using 'readtable'

```
DIdata = readtable('Dauphin_Island_hyd.csv');
```

You can open the table by clicking on the variable name in the workspace.
This is a bigger table, and we are interested in the water temperature and the depth, so find those columns and see what the column heading is. Remember you can call individual columns of a table by name, as

```
plot(DIdata.watertemp1_avg)
```

Note that the x-axis is just the index value, which is the data collected every 30 min.

To compare the data, we need to adjust the time values to be the same.
https://www.mathworks.com/help/matlab/date-and-time-operations.html

This is always a pain but the format for arcos is especially annoying – here's some janky code to turn it into MATLAB datetime format:

```
%fix time data
DIdata.timedata(DIdata.timedata == 30) = 2530; %not recognizing the 0
as an hour so make it 25 then fix below
DIdata.timedataS = num2str(DIdata.timedata); %turn into a string to
split the hour and minutes
DIdata.timedataM = str2num(DIdata.timedataS(:,3:4)); %minutes
DIdata.timedataH = str2num(DIdata.timedataS(:,1:2)); %hrs
DIdata.timedataH(DIdata.timedataH == 25) = 0; %turn the 25 back to 0
DIdata.DT =
datetime(DIdata.yeardata,01,DIdata.jday,DIdata.timedataH,DIdata.timeda
taM,0); %put in matlab datetime format
```

We would like to plot the arcos data on the same graph as the data we collected. This is easy to do once we get the datetime format working for both sets of data. You should have code that plots temperature on the left side of the y axis and depth on the right side of the y axis in one figure. To make sure you plot the next set of data in the same figure window, go back to your existing code and before the "plot" line, add a line of code that says

```
figure(1)
```

What this does is tells MATLAB to create a new figure in the "figure 1" window. You can then call that figure at any time later in your code by again typing

```
figure(1)
```

Now you can plot the data from your sensor on the same graph as the data from the arcos station – do this for temperature and for water depth. You'll need to do:

```
figure(1) %call back the fig 1 window
yyaxis left % specify to plot on the left axis where we plotted
temp
hold on % don't replace the existing graph, plot on top of it
```

When you plot your depth data (using *yyaxis right*), you'll note that our sensors are not at the same depth as the arcos instrument. It will be easier to compare the data if you adjust the depth on one of them. For our purposes, you can just pick a number to add or subtract from one or the other to get the signals closer to each other. Add a legend to your graph – look up *legend* in the MATLAB help.

2. Save your graph with your data and the arcos data plotted on the same graph. (5 pts)

3. Does your sensor track well with the weather station data? If your signals don't match, what do you think caused the differences? (4 pts)

4. You calculated the depth resolution of your pressure sensor already – what is the calculated resolution? Zoom in on your graph and see if you can pick out the resolution from your graph. Does it match what you expected? (5 pts)

Note: you can use the "+" and "-" magnifying glass buttons in the figure window just above the right side of the graph to zoom in and out. When you have two y-axes, this can be a little annoying; you need to click on the y-axis that you want to zoom on before zooming in. If it gets messed up, you can always click the "home" symbol to return the graph to the original zoom. You can pick numbers off the graph by holding the cursor on top of a spot on the line – a little window with x and y values should pop up. This only works if you are NOT in zoom mode. You can also use the command *grid on* to add a grid which can make picking out numbers a little easier.

It's a bit trickier to calculate the temperature resolution of the sensor, but we can do this by calculating the difference in temperature that would result from an increase of 1 arduino unit using the equations to convert Arduino reading to temperature. This is a fun exercise to do yourself, but if you're not experienced in MATLAB, it's a bit tricky. So I've done this for you in the attached MATLAB code (below).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%calculate resolution of temp sensor - varies with temperature
close all, clear

%set up constant variables
B = 3950;
T0 = 298.15;
R2 = 10000;

%convert sensor value to temperature using the Steinhart equation
sensorValue = 0:1:1023;
readVolt = sensorValue*5.00/1023; %convert arduino reading to voltage
thermRes = R2*((5.00./readVolt)-1); %convert voltage to resistance
```

```
tempInv = (1/T0)+(1/B)*log(thermRes/R2); %first part of Steinhart
equation
tempC = (1./tempInv) - 273.15; %2nd part of Steinhart equation

%calculate temperature difference from one sensor value to the next
tempErr = diff(tempC);

figure
plot(tempC(1:end-2),tempErr(1:end-1),'o')
xlabel('Temp C')
ylabel('Temperature resolution (deg C)')
grid on
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

5. Using this code and what you've learned about manipulating graphs in MATLAB, what temporal resolution do you expect to get from your temperature sensor? Zoom in on your data graph – what temperature resolution did you measure? Do these match? (6 pts)

OPTIONAL

If you'd like a little more practice with MATLAB:

A. The MATLAB default is to set the tick marks at midnight, which is not necessarily the best reference for our data. Look up the sunrise and sunset times and the predicted high and low tide times for the time period your instrument was deployed and plot these times as vertical lines on your graph (with some appropriate color scheme) using the command *xline*. (+1 bonus point)

B. In order to quantitatively compare our data to the arcos data, we need to adjust the data sets so they have the same sampling frequency. To do this, first smooth your data (google how to do this), then interpolate one data set to the same sampling frequency as the other (using the command *interp1* <- this is a number 1 not an L). Once you have your data sets aligned, plot the differences between your instrument and the arcos instrument as a function of time by subtracting one from the other. From here you can run statistics on how closely correlated the two instruments are (using *xcorr*) and determine the lag between them (e.g., if one instrument was further up the bay, we might expect to see a tidal lag in temperature). (+2 bonus points)

C. Download data for the same time period from another of the arcos weather stations – Cedar Point (CP) is a good option because it's not too far from DI so we would expect it to be fairly similar but possibly with a tidal lag. Add those data to your plot. How spatially variable are temperature and water height patterns in this area? (+1 bonus point)

PART 4.2: COMPARE DATA WITH CASTABLE INSTRUMENTS TO CTD CAST

STEP 1: PLOT YOUR DATA

Follow the steps you used when you deployed your sensor to import your data from your cast sampling (Step 1 of Part 4.1).

First plot your data to check to see if it looks okay. You can either plot your data against time like you did for the dock deployment, or, since we don't care about time here, you can plot the data against "scan count" or with no x-data (look up the MATLAB *plot* function to see how to do this).

6. a) Before you plot your data, think about and explain what it means for the data to look "okay" (Hint: To what depth did we do our cast? What temperature do you expect the water to be?). (3 pts)

   b) Plot the data and include your graph. Does it look "okay" based on your expectations? (4 pts)

We ultimately want to look at a profile of temperature as a function of water depth – First you will need to subsample the segment of data that includes the cast (we traditionally use just the downward part) and remove the excess data before and after your cast. You can do this in MATLAB by creating a new variable and using brackets to indicate rows and columns to subsample. The easiest way to do this is to zoom in on the graph and pick out the index values from the graph.

OPTIONAL. If you were going to be processing lots of these casts, you might want to automate this subsampling by writing code to find the beginning and end of the downcast automatically – can you figure out how to do that in MATLAB? Ask if you'd like help doing this!

Generally when we plot data, we plot the independent variable on the x-axis and the dependent variable on the y-axis. However, when looking at data collected as a function of depth, it can be easier to visualize the data if depth is plotted on the y-axis and the dependent variable is plotted on the x-axis. You want your graph to show the surface of the water (depth = 0 m) at the top and the deepest measurement on the bottom.

7. Next plot temperature as a function of depth with depth on the y-axis (and the surface of the water at the top of your graph) and temperature on the x-axis. Which are the independent and dependent variables? (7 pts)

STEP 2: PLOT THE CTD DATA

The CTD data are uploaded from the instrument using a proprietary software made by SeaBird, the manufacturer. The data are stored as a .hex file that requires a configuration file (.con) with information about the sensors on the specific CTD (i.e. calibration offsets) to open in the software. This step has been done for you, and the data have been output as a .cnv file, which is essentially a text file including all of the header or metadata information (i.e., rows of words telling you about the data) followed by the data for each variable that is recorded at each time point (i.e., rows of numbers).

OPTIONAL. If you are interested in learning how to use the SeaBird software to interface with the CTD, please ask! We are skipping this step due to time constraints, but if you regularly use or expect to use CTD data, it is worth understanding where the data come from!

Our first task is to get these data into MATLAB in an organized way (a MATLAB table with columns for each type of data coming from the different sensors on the CTD) so we can then plot the data. To do this, we need to be able to tell MATLAB the following information:

- Where the numbers we want are (the rows and columns that should be included in the table)
- How the numbers are separated – this is called "column delimiters"
- What each column of numbers is – the column or variable name
- What kind of variable each column contains – most of our data are "numbers" but we could also have categorical data, text data, etc.

We will use the Import Wizard to import our data.
https://www.mathworks.com/help/matlab/ref/importtool.html
https://www.mathworks.com/help/matlab/import_export/import-data-interactively.html

It's a good idea to open the raw .cnv file in another program so you can see it (excel or text edit). Read through the header information, looking for the names of the columns of data, noting the units for each one as well as the range (this will help you make sure you are matching the right name to the right column). This is a bit challenging, so take your time and try to understand what each of the different options in the import wizard does.

Once you get your data looking right in the import wizard, click on "Import Selection" and generate a script – that way you can import the data for multiple CTD scans using the code.

The next step is to subsample the data from the downward cast -

8. To do this, plot the depth, with the index value or scan count on the x-axis. Based on how we deployed the CTD, what should this plot look like? Show your plot and explain (briefly) how what we did corresponds to different parts of the plot (i.e., annotate the plot) and whether it looks correct. (6 pts)

Create a new variable that subsamples just the downcast – you can do this by picking out the scant count values from your plot (you might want to turn *grid on*) or (more advanced) you can write a code that will automate the process by picking out the starting and ending points of the cast.

9. You can now plot your data as a function of depth. Plot the following variables with the surface (depth = 0 m) at the top of your plot:
    - Temperature
    - Salinity
    - Density
    - Oxygen
    - Fluorescence
    - PAR/Irradiance
    - Beam attenuation

For all graphs, label axes and be sure to include units, which you can get from the header information. Explain what each graph shows – do these variables increase or decrease with depth? Are there sharp transitions? What do these graphs tell you about the ocean when we took the CTD cast (i.e., what are these variables and why might we measure them?)? (10 pts)

[Note: you can plot these data on the same graph but it's a bit tricky since the range of values differs among the variables. It's fine to just plot each variable separately for this assignment, or if you want to plot them together you can use *subplot* or *tiledlayout* or *yyaxis*. If any of these variables are not included, just make a note in your lab report that you don't have those data.]

10. Next compare your instrument data to the CTD data by plotting temperature from both instruments as a function of depth in the same graph (in MATLAB, use the command *hold on* to plot graphs on the same axes). Use colors to distinguish the data from your instrument from the CTD data (look up *plot* in MATLAB). How closely did your data correspond to the CTD data? Do you see a thermocline in the data? Does the depth of the thermocline match for the two instruments? (10 pts)

11. Look closely at the sampling resolution by plotting the data points as '.' on top of the line graph (look up the command *plot*). What was the approximate depth resolution of your instrument? What was the approximate depth resolution of the CTD? (8 pts)

PART 4.3: LESSONS LEARNED AND NEXT STEPS

12. Experimental design is an iterative process of designing and testing until you reach a point where it's good enough to get the data you need. We just completed the steps of building a prototype instrument to measure and log temperature and depth. If your goal is to record these data off the DISL dock for a week, is your instrument good enough right now or does it need further modification and another test run? Explain your answer. If it needs further modification, what do you need to do? (9 pts)

13. How did your instrument perform in a castable form? Why do oceanographers buy CTD instruments for ~$35K rather than use DIY instruments like we built (which cost <<< $35K)? (9 pts)

14. What is the most valuable thing you learned from this project? (9 pts)